A Framework for the Competitive Evaluation of Model Inference Techniques

Neil Walkinshaw
Department of Computer Science
The University of Sheffield
Sheffield, UK
nw@dcs.shef.ac.uk

Kirill Bogdanov
Department of Computer Science
The University of Sheffield
Sheffield, UK
kirill@dcs.shef.ac.uk

Christophe Damas
Département d'Ingénierie Informatique
Université Catholique de Louvain (UCL)
Louvain-la-Neuve, Belgium
christophe.damas@uclouvain.be

Bernard Lambeau
Département d'Ingénierie Informatique
Université Catholique de Louvain (UCL)
Louvain-la-Neuve, Belgium
blambeau@gmail.com

Pierre Dupont
Département d'Ingénierie Informatique
Université Catholique de Louvain (UCL)
Louvain-la-Neuve, Belgium
pierre.dupont@uclouvain.be

ABSTRACT

This paper describes the STAMINA competition[1], which is designed to drive the evaluation and improvement of software model-inference approaches. To this end, the target models have certain characteristics that tend to appear in software-models; they have large alphabets, and states are not evenly connected by transitions (as has been the case in previous similar competitions). The paper describes the set-up of the competition that extends previous similar competitions in the field of regular grammar inference. However, this competition focusses on target models that are characteristic of software systems, and features a suitably adapted protocol for the generation of training and testing samples. Besides providing details of the competition itself, it also discusses how outcomes from the competition will be used to gain broader insights into the relative accuracy and efficiency of competing techniques.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*elicitation methods*

General Terms

Design, Documentation, Experimentation

[1]http://stamina.chefbe.net/

1. INTRODUCTION

In practical software development, models that specify the intended behaviour of the system are often omitted because of the time and cost involved in their generation and maintenance. This has led to the development of a multitude of different model-inference techniques, which aim to address this problem by inferring models automatically [4, 7, 19, 1, 15, 16, 8, 9, 27]. These tend to operate by taking samples of known software behaviour (either from scenarios supplied by the developer, or traces of actual program executions) and aim to return a model that is sufficiently general to accurately encapsulate the complete behaviour of the system in question.

Inference techniques are difficult to compare to each other. In the absence of any suitable benchmarks, their published evaluations revolve around differing sets of software models. The selected models are however rarely diverse enough to enable a reliable, systematic comparison of the techniques; they may be only of a specific size, or have a particular number of labels etc. The measurements that are used for evaluation often vary from technique to technique [28, 26]. Some measurements concentrate on accuracy, whereas others concentrate on efficiency, and both factors are often measured in different terms. Replicating experiments is challenging because implementations are not always available, and can be time-consuming to re-implement if the inference process in question is complex.

This paper describes a competition framework that has recently been deployed with the aim of addressing the above problems. STAMINA is a competition for the comparison of state machine inference approaches. The intention is that it will provide a current snapshot of the best performing inference techniques, and spur the development of techniques that improve on the state of the art. It also serves as a means for collecting data that can be used for a more detailed analysis, to provide insights into the circumstances under which particular approaches excel or fail. Although the formal competition will only run for a relatively short amount of time, the infrastructure will be openly available for the longer term, as a benchmark for the evaluation of new techniques.

The rest of the paper is structured as follows. Section 2 lays out the background, providing details of the models (labeled transition systems), and the passive model-inference paradigm. Section 3 provides an overview of the competition framework. Section 4 provides details on the base materials for the competition – it shows how the target models are synthesised, along with how the training and test sets are generated. Section 5 discusses how the data that is submitted can be used as the basis for a comprehensive empirical comparison of the competing techniques. Section 6 discusses related work, and section 7 looks at future work and provides some conclusions.

2. BACKGROUND

A brief introduction is provided to the essential notations of DFA's and their languages. This is followed by an introduction to the passive model inference setting. The section finishes by describing the challenge of reliably evaluating and comparing different passive model inference techniques to each other.

2.1 Definitions and Notation

The STAMINA competition revolves around the inference of *Deterministic Finite Automata* (DFA's). The accuracy of the inferred DFA's is established with respect to the similarity of its language to that of the target model. The definitions of a DFA and the language of a DFA are presented below.

Definition 1. A Deterministic Finite Automaton is a quintuple $(Q, \Sigma, \Delta, F, q_0)$, where Q is a finite set of states, Σ is a finite alphabet, $\Delta : Q \times \Sigma \rightarrow Q$ is a partial function and $q_0 \in Q$. F is a subset of Q and represents the set of final (accepting) states where the system may terminate. A DFA can be visualised as a directed graph, where states are the nodes, and transitions are the edges between them, labelled by their respective alphabet elements.

It is worth noting that software systems are often modelled with a closely related formalism: The Labelled Transition System (LTS). This is effectively a DFA where $F = Q$. These can be used to model a subset of the languages that can be modelled by DFA's, where every prefix of an accepting sequence is also accepted (this is known as the set of *prefix-closed* languages).

When referring to different DFA's X and Y, the use of the subscript (e.g. Q_X) is used to refer to an LTS element for that specific machine.

To define the language of a DFA, we draw on the inductive definition for an extended transition function $\hat{\delta}$ used by Hopcroft *et al.* [11]. For a state p and a string w, the extended transition function $\hat{\delta}$ returns the state q that is reached when starting in state p and processing sequence w. For the base case $\hat{\delta}(q, \epsilon) = q$. For the inductive case, let w be of the form xa, where a is the last element, and x is the prefix. Then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$.

Definition 2. Given the extended transition function, the language of a DFA A can be defined as follows:
$L(A) = \{w | \hat{\delta}(q_0, w) \in F\}$. The complement of a language L (i.e. the set of sequences that do not belong to L) is denoted $L(A)^C$.

2.2 The Model Inference Setting

Model inference techniques can be broadly divided into two categories: active techniques (which assume the presence of some oracle to provide feedback) and passive techniques. The purpose of this competition is to compare techniques in the latter category.

In this inference setting, the assumption is that the technique starts with a sample of the behaviour of the target software system, and that the behaviour can be characterised by some (hidden) DFA A. The sample is in the form of a set of sequences in Σ^*. In the software-engineering domain these may be program traces, or scenarios that have been supplied by the developer. The sample is supplied in two sets of sequences; one set S^+ is known to belong to $L(A)$, and the other set S^- is known to belong to $L(A)^C$.

When combined, the contents of S^+ and S^- will not necessarily be sufficiently diverse to guarantee that an accurate model will be inferred. The notion of a “complete” sample varies from technique to technique. The de facto notion of a complete sample was established by Oncina *et al.* in their work on the Regular Positive Negative Inference (RPNI) algorithm [18]. They showed that, to ensure an accurate result, the sets $S^+ \cup S^-$ would have to be structurally complete and *characteristic* of the target model [10]. In other words, they would need to contain a sufficiently diverse set of samples to (a) cover every state transition and (b) contain enough information in S^- to distinguish every pair of non-equivalent states. The problem is that characteristic samples tend to be vast, whereas in a practical software-engineering setting, it tends to be the case that we have only a small subset of program execution traces or scenarios. The challenge is to generate a model that is reasonably accurate, even if the given set of samples is sparse (non-characteristic).

Numerous techniques have been developed to infer state machines in this setting. These include Markov model and Neural net-based approaches [7], variants of an inference known as k-tails [4, 7, 19, 1, 15, 16], and variants of the RPNI [8, 9, 27]. Many of the algorithms that are used to infer software DFA's originated in a closely related sub-field of Machine Learning, known as *regular grammar inference*. A regular grammar can be represented as a DFA, and algorithms such as $k - tails$ [3] and RPNI [18] were originally developed in this context.

2.3 The Evaluation and Comparison of Inference Techniques

Empirically evaluating the performance of model inference techniques is challenging. Their performance can be affected by a number of factors. The size and complexity of the target machine are of obvious significance. The number of states and transitions, and the size of Σ are indicators of this complexity, as is the depth (the longest "shortest walk" from the initial state q_0 to any other state). Besides the target model characteristics, the sparsity of the training set is another key factor; the more complete the training set is, the easier it is to infer the states and transitions in the final model. This explains the difficulty in reliably assessing the performance of a technique. All of these variables need to be controlled, to ensure that the experiment is not unfairly biased (either for the better or worse).

Within the domain of software-engineering, model inference techniques are usually evaluated on a case-study basis (e.g. Lo *et al.* use a CVS system [15], Walkinshaw *et al.* use a drawing tool [27], and Damas *et al.* use a water pump controller [8]). A real software system is selected as the target for the inference technique, and is used as the basis for eliciting a set of traces / scenarios. This serves the purpose of establishing the feasibility of the technique, but fails to provide any reliable basis for comparison with other techniques.

In the Grammar Inference domain, this problem has been addressed by organising competitions. Competitions offer a means to compare different techniques with respect to the same set of target models, with the same set of samples. Most competitions follow the example set by the successful Abbadingo-One competition [13]. The organisers generate a set of quasi-random target models, but the competitors are only given a training set, and a "test set". Once the competitors have inferred a model with the training set, they classify each string in the test set as either belonging or not belonging to their hypothesis model. This results in a binary string, which they submit to the competition web server. If their classifications are at least 99% correct, they are deemed to have successfully inferred the model. Different categories are established that correspond to different levels of difficulty; the techniques to correctly infer the most difficult models are selected as winners. The Abbadingo-One competition produced the Blue-Fringe algorithm by Price, which is deemed to be the current benchmark for grammar inference algorithms.

Unfortunately, competitions such as Abbadingo-One are unsuitable for the evaluation of software model-inference techniques. The problem primarily lies in the way the target machines are generated. The alphabets used to generate the random machines are only binary ($\Sigma = \{a, b\}$), meaning that each state can have at most two outgoing edges. For a state machine target size of n, a random directed graph (representing the transition structure of the machine) is generated on $\frac{5}{4}n$ nodes. The edges between states are inserted at random, and then a sub-machine that is reachable from the initial state is selected. This results in a set of machines with roughly the same depth and size.

These machines are however not at all representative of typical software models. DFA's that model software systems involve state transitions that may be triggered by any of a large number of events (mouse-clicks, function names, IO events, etc.). The size of the alphabet, and the number of outgoing transitions from a given state can be very large, and vary significantly from state to state. The previously described random graph generation algorithm produces a homogeneous network, where any pair of states are equally likely to be connected. In practice, software DFA's do not obey this law; certain states represent *"hub"* states, e.g. the root of a menu system, or a defacto error-handling state, whereas other states may be intermediate parts of a longer sequence of specific operations.

Generating random training and testing samples for past competitions has been straightforward. The approach for competitions such as Abbadingo has been to simply generate random binary sequences up to a prescribed length, and to subsequently use the target model to classify them as "accept" or "reject". However, doing so with machines that have a larger alphabet is no longer viable; an overwhelming majority of random sequences in Σ^* is likely to be classified as "reject", and of the few sequences that are accepted, it is unlikely that they would provide any useful coverage of the target machine. In this situation, if 99% of sequences were correctly classified (mostly as negatives), it would say very little about the actual accuracy of the inferred model [28].

3. THE STAMINA FRAMEWORK

The STAMINA competition is based on the inference of a set of 100 random DFA's, each of which is roughly 50 states in size. For each machine, competitors are only given a training set, which is a sample of sequences, where each sequence is classified according to whether it is accepted or not by the target machine. Having used the sample to infer a hypothesis model, competitors can also download a test set (an unclassified sample that is different to the training set), and they have to classify the test set on their hypothesis model. The solution is submitted as a binary string, where a '1' represents a test that is accepted by the hypothesis, and a '0' represents a test that is not accepted.

The challenges vary in difficulty, depending on the size of the alphabet of the target DFA, as well as the sparsity of the provided training sample. The various levels of difficulties are depicted in the column and row labels of table 1 (each cell represents five of the target models, the generation of which is discussed later). The size of the alphabet is either 2, 5, 10, 20 or 50. The sparsity of the sample ranges from 100% to 50%, 25% and 12.5%. The full set of sequences at 100% has been empirically selected to ensure that the base-line Blue-Fringe technique produces a reasonably accurate result for the simplest problems ($|\Sigma| = 2$, sparsity = 100%) without solving the cell[2]. Thus, any solution that manages to solve the cell will have to out-perform the baseline technique.

3.1 Measuring the Accuracy of a single Hypothesis Model

Solutions are submitted as a binary string, representing a sequence of tests that are accepted or rejected by the hypothesis machine. To establish the accuracy, it is compared to a reference string representing the correct classifications of the test set on the target model. The overlap between the two binary strings is measured with the *Balanced Classification Rate (BCR)*. The Harmonic BCR measure is chosen

[2]Although the average BCR score for cell $|\Sigma| = 2, 100\%$ is 0.99, the Blue-Fringe did not achieve this for every target model, and so did not successfully solve the cell.

	Sparsity			
$\|\Sigma\|$	**100%**	**50%**	**25%**	**12.5%**
2	0.99 (1)	0.95 (1)	0.67 (3)	0.66 (3)
5	0.97 (1)	0.78 (2)	0.59 (4)	0.52 (4)
10	0.93 (1)	0.64 (3)	0.51 (4)	0.5 (4)
20	0.91 (1)	0.63 (3)	0.54 (4)	0.51 (4)
50	0.81 (2)	0.64 (3)	0.57 (4)	0.5 (4)

Table 1: Table with mean BCR results for the five challenges in each cell, as computed by the baseline Blue-Fringe technique. Scores are associated with a difficulty grade from 1 (easiest) to 4 (hardest)

Difficulty level	Score
1	$0.9 \leq score \leq 1$
2	$0.7 \leq score < 0.9$
3	$0.6 \leq score < 0.7$
4	$0 \leq score < 0.6$

Table 2: Calibrating the mean BCR scores for a given cell, based on the scores of the benchmark Blue-Fringe algorithm shown in table 1

because it places an equal emphasis on the accuracy of an inferred model in terms of its acceptance of positive sequences, as well as its rejection of sequences that should be rejected. It also does not require the test set to be balanced in terms of its positive / negative sequences.

Harmonic BCR combines two factors. *Sensitivity* is the proportion of positive matches that are predicted to be positive. So in terms of the sets true positives (TP) and false positives (FP), $Sensitivity = \frac{|TP|}{|TP \cup FN|}$. $Specificity$ is the proportion of true negatives that are predicted to be negative, so $Specificity = \frac{|TN|}{|TN \cup FP|}$. Harmonic BCR[3] is the harmonic mean of the two:

$$BCR = \frac{2 * Sensitivity * Specificity}{Sensitivity + Specificity}$$

3.2 Scoring the Overall Performance of a Technique

The overall performance of a technique is measured in terms of the 'cells' in table 1 that are solved. Each cell represents a different difficulty-level, and is solved by accurately inferring five randomly generated target models (random model and sample generation are discussed in the next section). A model is deemed to have been accurately inferred if the BCR score is greater than or equal to 0.99. If the score is any less than this, the competitor does not get to find out the score, in order to prevent the use of feed-back to home-in on the correct result.

Since cells vary in difficulty, the scores that are awarded to a competitor for solving a cell must vary accordingly. The scores are calibrated by running the best-known passive inference algorithm on the training samples, and establishing the accuracy of its outputs. The cell-values in table 1 show the average score of the Blue-Fringe algorithm [13] on the five models. These are categorised into four levels from easy (1) to hard (4), which are the points attributed to a technique for each cell solved according to the scheme in table 2. The winning technique will be the first one to solve a cell in the hardest category.

[3]Conventionally, BCR is simply computed as the arithmetic mean of sensitivity and specificity, but the harmonic mean is preferred here because it favours balance between the two.

3.3 Running the Competition

The competition is run as a web-server, which is implemented in Ruby on Rails. All of the 100 problems are openly available to download. Competitors can log-in to submit their solutions. The page incorporates a dynamic league-table, that shows a real-time list of the winning techniques.

There are three incentives to encourage participation. (1) To ease participation, the source code for the current baseline solution has been made openly-available (discussed below). (2) As an incentive, a financial reward of £700 (∼$1080) is offered to the winners. (3) As a further incentive, the authors of the best techniques will also be invited to describe these in a special issue of the Journal of Empirical Software Engineering.

Developing an implementation for a state machine inference technique is a time-consuming process. This is one of the main hindrances to potential participants. To ease this, a well-documented version of the current baseline technique (the Blue-Fringe EDSM algorithm [13]) has been made openly available on the website. This is the implementation that has been used to calibrate the scores shown in table 2, and represents the best-known solution to the passive inference problem.

4. SYNTHESIS OF TARGET MODELS, TRAINING AND TESTING SAMPLES

This section describes the algorithms that were used to synthesise quasi-random DFA's that are more representative of typical software models. As mentioned above, this means that the traditional approaches for generating training and testing samples are no longer applicable, so an alternative algorithm is described.

4.1 Target Machines

The task of generating a random DFA is not trivial. Conventional algorithms (e.g. as used in previous grammar inference competitions [13]) generate transition structures that are homogeneous; there is a uniform probability that any pair of states is connected by a transition. In reality this is not the case – in software systems certain states play a more central role than others. Consequently, for this competition we have developed a new random state machine generation algorithm that attempts to produce random machines that are more realistic. To gain an insight into the key characteristic of software state machines, a study was carried out for existing state machine models. The study is discussed in section 4.1.1. This is followed by a description of the random DFA algorithm itself.

4.1.1 Observed Software model characteristics

A random DFA generation algorithm was developed with the aim of producing DFA's that are representative of software models. To gain an overview of what constitutes a "typical" software model, a sample of 20 systems (mostly from RTPs and research publications) was analysed (the state transition structures have been made available [4], and give an idea of the nature of software models considered in this work). Although the sample is too small to form any authoritative conclusions, findings can be interpreted as being indicative. The DFA's were analysed in terms of their states, transitions and alphabet size, as well as their distributions of in-/out-degrees and depth. The main results, which informed out design of the random DFA generation algorithm, are briefly listed below.

1. **Alphabet size.** There was no relationship between the size of an alphabet and structural features, such as the number of states / transitions, or the depth of the DFA.

2. **Relationship between depth and states.** There was a strong relationship between the depth and number of states, which is roughly modelled by the following relationship (as established by linear regression): $depth = (0.36 * states) + 1.3$.

3. **Degree distributions.** No straightforward distribution was found in terms of the number of in-/out-degrees per state. However, by using a more global measure, a trend was identified. Most states would have 1-2 in-going and out-going edges, however there would usually be 1-2 states that had either a high "hub" score or a high "authority" score. "Hub" and "authority" scores are computed by Kleinberg's HITS algorithm [12]. These are best explained in their intended context, where vertices represent web-pages. The intuition is that a web page has a high "authority" score if it is the target of many links (has a high in-degree), and it is linked to by pages that contain links to other pages with high authority scores. A page has a high "hub" score if it links to lots of authoritative pages. As implied by this definition, the computation of the two scores is a recursive process, with one affecting the outcome of the other. In the DFA context, state with a high "hub" score can be interpreted as a root state, which is the starting point for a range of different sequences of events, and a state with a high "authority" score can be interpreted as the ultimate target of lots of different state trajectories through the DFA.

4.1.2 Random DFA Algorithm

Generating a "realistic" random state machine is not as straightforward as it may seem. There are certain constraints that must hold; every state must be reachable from some initial state, it must be deterministic and minimal. On top of that, it must be possible to generate a population of those machines that obey the three characteristics listed above.

Observation (3) from above notes that state machines tend to only have a small number of hubs and authorities. Graphs of the World Wide Web (known as "complex networks"), which spurred algorithms such as HITS, have also been shown to have only a small proportion of Hubs and Authorities with high scores. Although the state machines we have studied differ in both scale and nature from conventional complex networks, this observation implies that algorithms for the generation of random complex networks make a reasonable starting point for the generation of random state machines.

[4]http://www.dcs.shef.ac.uk/ nw/stamina/

Numerous algorithms have been developed to synthesise random complex networks. The forest-fire algorithm by Leskovec *et al.* [14] has been shown to produce directed graphs that are especially suited for representing complex networks in a variety of domains. It is based on an iterative algorithm, where a new node is added at each iteration, and new edges are added to existing nodes by partially traversing the existing edges that link them together. The rest of this section provides a brief overview of the forest fire algorithm, and shows how it has be adapted to produce state machines.

This description closely follows that of Leskovec *et al.*, who can be referred to for further details. The algorithm has three parameters: a *forward-burning probability* f, a *backward-burning ratio* b, and the number of vertices n. Consider a node v to be added to the graph at a time t where $0 < t \leq n$. Node v forms an edge to nodes in the graph at time t as follows:

1. Choose a random ambassador node $w \neq v$ and form an edge $v \to w$.

2. Generate two random numbers x and y that are geometrically distributed with means $f/(1-f)$ and $fb/(1-fb)$. Node v selects x in-edges and y out-edges of w to nodes that are not yet visited. If there are not enough nodes available, it selects as many as it can.

3. v forms out-edges to the end-points of the selected edges from and to w and applies step (2) recursively for each of those nodes. As the process continues, nodes cannot be revisited.

The above algorithm cannot be used as-is to synthesise state-machines. As new nodes are added, they are unreachable from any of the other nodes in the machine. By default, an edge cannot be added from one node to itself, ruling out self-looping states, which are common in software models. The original algorithm does not account for the notion of terminal states. Furthermore, a strategy is required to ensure that the state transitions in the final machine are suitably labelled (i.e. that the machine is deterministic and minimal). This needs to account for the fact that there could be multiple transitions between the same pair of states.

The following adaptations have been made to ensure that the final graph is state-machine like.

- **Alphabet:** To add the transition labels, an additional parameter a is used, which is the upper limit on the size of a vector of numbers representing different elements of the alphabet. Every time an edge is added, it is labelled with a random element from that vector. The possible choices are curtailed to ensure that a selected element will not cause the machine to be non-deterministic. If there are no available alphabet elements left, the edge is not added.

- **State reachability:** To ensure that each state is reachable, every time a state is added, instead of connecting an edge from the new state to an ambassador state, the reverse edge is added (from the ambassador to the new state).

- **Accepting states:** Every time a state is added it is randomly labelled (with a probability of 0.5) as an accepting (final) state or not.

- **Self-loops:** In step 2 of the conventional algorithm, it is impossible to add edges from a state to itself. We have added a parameter s to make it possible to specify the probability for this to occur.

To identify suitable parameter values for the algorithm, random DFA's were synthesised for a range of parameters f, b and the self-looping probability s. The alphabet-size parameter a only affects the structure of the machine for small alphabet sizes (i.e. $a = 2$), constraining states to an out-degree of 2. However, since this is a special case it is not factored in to the calibration of the main parameters. For every configuration, certain measurements were plotted, including the number of transitions, states, depths, and hub/authority distributions. These were compared to the equivalent plots from the real sample of machines. Ultimately, the judgement of which configuration to choose is to a large extent qualitative. Several configurations produced reasonable looking DFA's, but the configuration $f = 0.31$, $b = 0.385$ and $s = 0.2$ resulted in measurements that were deemed to be most suitable in terms of the plots, as well as a manual inspection of the DFA's themselves.

4.2 Training and Testing Samples

As stated in the background, the algorithms that are used to generate random samples for previous competitions are no longer suitable for DFA's with large alphabets, because they tend to generate too many negative sequences. The sampling procedure used in this competition computes direct walks over the target machine as opposed to randomly combining elements of the alphabet. The procedure can be summarized as follows:

1. Using a random walk algorithm (see details below) a first sample is generated from the target automaton. The sample contains exactly 20000 sequences but may contain duplicates. The random walk algorithm is tuned to provide an even balance of sequences that do and do not end in a terminal state.

2. The sequences are equally partitioned in two disjoint sets (retaining the terminating / non-terminating balance). One is designated as the training pool, and the other as the test pool. Any duplicate sequences are removed from the test pool.

3. The final testing sample is computed by randomly selecting a maximum of 1500 sequences from the testing pool.

4. The training sample generation depends on how *sparse* the sample is supposed to be. As was explained in section 2, the difficulty of an inference problem is determined not only by the characteristics of the target machine, but also by the completeness of the given sample. In our competition, we incorporate four levels of sparsity: 100% is a notionally complete sample, and 50%, 25%, 12.5% are subsets. If we denote the required sample sparsity for the problem at hand as p, the learning sample is created by randomly selecting $0.9 * p$ percent of training pool generated in (2).

The different parameters above (20000 strings initially generated, 1500 strings maximum in the test set and the 0.9 multiplication factor for the learning sample) have all been calibrated against the performance of the current baseline inference approach (see section 3.3). They were chosen to ensure that the baseline performs well on the simplest challenges (at 100% with an alphabet of 2), but does not manage to achieve a score high enough to solve it (the scoring is discussed in detail in section 3).

4.2.1 *Random walk algorithm*

A dedicated random walk algorithm has been implemented to generate positive and negative sequences of the initial sample described above. It generates positive sequences by walking the automaton from the initial state, randomly selecting outgoing state transitions with a uniform distribution. When a non-terminal state v is reached, the generation ends with a probability of $1.0/(1 + 2 * outdegree(v))$. The length distribution of the sequences generated that way is approximately $5 + depth(automaton)$, ensuring a high probability of generating structural complete samples (this is inspired by the Abbadingo competition [13]).

Negative sequences are generated by editing positive strings obtained as above. Three kinds of edits are used: substituting / inserting / deleting a symbol. In all cases, the edit-location is chosen from an uniform distribution on the sequence length. The number of edits is chosen with a Poisson distribution centred on 3, and the edit kind with a uniform distribution. The sequence is simply discarded if the edited version still ends in an accepting state.

5. EMPIRICAL ANALYSIS

There are two objectives for this competition: (1) to identify the best inference technique, and (2) to obtain more detailed insights into the circumstances under which different techniques excel or flounder. Establishing the best inference technique is reasonably straightforward, and the means by which this is achieved have been outlined in section 3. However, from an academic standpoint, the second task is more interesting. The competition server will store the test results that are returned for every technique. This section describes how these can be analysed, to provide insights into how techniques compare to each other, and why one might excel against another.

Figure 1 illustrates the structure of the data that are available for analysis. Each entry consists of up to 100 solutions (binary strings of test results for the 100 possible problems), depending on how many cells have been attempted. Every

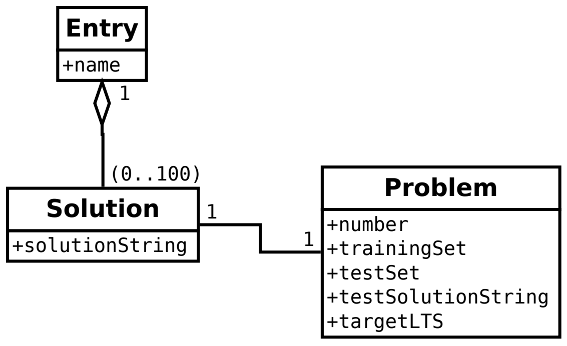

Figure 1: Relationships between data available for analysis

solution is associated with its respective problem. The difficulty of the problem – the sparsity of the training set and the size of the alphabet – can be identified via its number (every cell in the table is linked to five problem numbers). The training and test sets are the only elements of the problem that are visible to the competitor. Besides that, two further elements will figure in the analyses. The test solution string is the binary string that contains the correct solutions for the test set, and the DFA is the actual transition system of the target.

These data collectively permit a more in-depth analysis of the strengths and weaknesses of individual techniques. They also enable different competing techniques to be compared with each other. For example, if two techniques perform similarly in the context of the competition, it becomes possible to see whether they are performing similarly on individual problems, or whether they might have complementary strengths and weaknesses.

5.1 Measuring Classification Accuracy

The main goal will be to analyse the performance of a technique with respect to its classification of tests. This analysis can be carried out in two respects: (1) the accuracy of a technique with respect to the target model (comparing `solutionString` to `testSolutionString`), or (2) establishing the similarity of two competing techniques (comparing instances of `solutionString` from different entries).

Whereas the competition uses the Balanced Classification Rate, there are numerous other measures for the comparison of binary strings. All measures are computed in terms of the basic elements of the confusion matrix: the set of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). The metrics that can be computed include summary measures that can be used to interpret the general performance of a technique, such as Accuracy, BCR, and the F-measure. They also include more detailed measures that focus a particular aspect such as Precision, Recall and Specificity. The details of how these are computed are presented in Sokolova and Lapalme's overview [20].

When analysing the performance of a single entry, these measures can provide insights into why a technique was successful. The various accuracy measures described above, when coupled with the difficulty of the challenge (sparsity and alphabet size), can be used as the basis for a comprehensive performance profile for the technique. It is possible to find out more detailed characteristics of the performance. For example, a technique might tend to produce DFA's that are too general (low Specificity and high Sensitivity) for problems with small alphabets; this would become immediately apparent with the above measurements.

If two competing techniques are of interest, the measures could be used to compare their individual solutions. Any of the summary measures can be used to establish whether they are broadly similar. In cases where there is disparity, measures such as Precision and Recall can provide insights into where they differ, and whether one might be complementary to the other.

5.2 Mapping Classification Results to the Target DFA

The analyses discussed above only account for the overlaps between binary strings, such as the overlap between classification of test sequences and their expected solutions. This provides insights into the extent to which the languages of the target and inferred DFA's overlap (see section 2 for the definition of the language of an DFA). However, they provide no insight into the extent to which structural elements of the DFA are inferred – i.e. whether or not an approach has especially successful or unsuccessful at inferring particular states or transitions.

It is possible to gain an impression of which elements have been inferred by the combined analysis of `solutionString`, `testSet` and `targetDFA`. By combining `solutionString` and `testSet`, we know which tests are accepted and rejected by the inferred DFA. These accepted / rejected test cases can be traced in the `targetDFA`; states and transitions that do not conflict with any of the test results from the inferred DFA can be deemed to be properly inferred. Any conflicts indicate a mistake in the inferred model.

Knowledge of which parts of the machine have been (mis-)inferred complements the language-based measures discussed above. Correctly or incorrectly inferred states and transitions can be used to explain results such as poor precision or recall. Different inference techniques can be compared to each other, with respect to their ability at inferring particular areas of the target DFA.

6. RELATED WORK

As mentioned in section 2, competitions have been used to spur the development of novel techniques in numerous research areas, mostly centred around the broad field of artificial intelligence and machine learning. One popular example from the field of bioinformatics is the Critical Assessment of Techniques for Protein Structure Prediction (CASP) competition series [25], a biennial competition series that has become one of the main drivers for the development of protein structure prediction techniques. In the field of automated reasoning, the CADE ATP System Competition (CASC) [24] has been a successful series of competitions to compare the performances of automated theorem provers. Because of the close relationship between state machine inference and grammar inference, the rest of this section focusses on those competitions.

6.1 Regular Grammar Inference Competitions

The authors are aware of four competitions to infer regular grammars or deterministic DFA's. Three of these are concerned with passive model inference, and one is concerned with active inference techniques.

The Abbadingo-One competition [13] took place in 1998, and gave rise to the EDSM / Blue-Fringe algorithm, which is used as the baseline technique in our competition. This was followed by the Gowachin noisy DFA inference competition in 2004, which was followed up by the similar GECCO 2004 noisy DFA inference competition [17]. Their processes are broadly as described in section 2. The Gowachin and the GECCO competition however involve a step that adds noise to the training data. As stated in the background, STAMINA differs from these competitions in three ways: (1) the target models have larger alphabets, (2) the testing and training samples are derived directly from the target model, and (3) results are evaluated with the BCR metric as opposed to simply counting the proportion of correct classifications.

The ZULU competition [6][5] is an ongoing competition for the inference of DFA's in the presence of an oracle. The competition is geared towards the development of improved version of Angluin's L^* algorithm [2], which is developed for the setting where, besides an initial training sample, the learner can ask membership and equivalence queries from an oracle (though the competition does not account for equivalence queries). Besides the availability of an oracle, our competition differs from ZULU in the manner the target models are generated (it uses the same algorithm as the other regular inference competitions).

6.2 Non-Regular Grammar Inference Competitions

The Omphalos competition was run in 2004 (see work by Starkie *et al.* [21, 22]) to spur the development and comparison of techniques to infer context-free grammars. This form of grammar presents a more challenging basis for model-inference, and generally accepted to be intractable. Unlike regular grammars, context-free grammars are not represented by DFA's, but by production rules. Generating a competition for the inference of such grammars is particularly challenging. Unlike regular grammars, it is difficult to compare CF-grammars to each other in terms of their complexity and difficulty. Furthermore, it is difficult to gauge when a training or testing sample is notionally "complete".

The Tenjinno competition was run in 2006 [23], and was concerned with the inference of transducers. A transducer is a formalism that can (under certain circumstances) be interpreted as an non-deterministic finite automaton, but instead of labelling each transition with an element from Σ, it is labelled with a translation $from \rightarrow to$. So a walk through the machine represents the translation of one sequence $< from_0, \ldots, from_i >$ to a corresponding sequence $< to_0, \ldots, to_i >$. As with the Omphalos competition, the increased complexity of the transducer model required new approaches to training and evaluating hypotheses.

[5]http://labh-curien.univ-st-etienne.fr/zulu/

7. CONCLUSIONS AND FUTURE WORK

The competition server has been set up, and is accepting submissions [6]. It is scheduled to end at the beginning of 2011. There are several important benefits to be gained. Ultimately, a winning entry will further the state-of-the-art, providing a more suitable basis for the passive inference of software models. However, even if there is no dominant technique, we will still obtain insights into the relative strengths and weaknesses of competing techniques. It is envisaged that this more detailed knowledge will point to new potential avenues of research for the development of improved inference approaches.

The aim is to establish this competition as a regular event. There are several elements that could be changed for future competitions. The current evaluation of techniques is based on a language-perspective. However, if competitors were to submit the actual inferred DFA's instead of test solutions, it would be possible to produce a much more in-depth evaluation (e.g. by comparing the structure of the inferred DFA to the target [5]). Another potential change would be to introduce an oracle that is capable of answering queries (in a similar vein to the ZULU competition).

8. ACKNOWLEDGEMENTS

Work on the competition is funded or partially funded by: the EPSRC STAMINA project (EP/H002456/1), the EPSRC REGI project (EP/F065825/1), the Regional Government of Wallonia (GISELE project, RW Conv. 616425) and the MoVES project (PAI program of the Belgian government).

9. REFERENCES

[1] G. Ammons, R. Bodík, and J. Larus. Mining Specifications. In *29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 4–16, Portland, Oregon, 2002.

[2] D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75:87–106, 1987.

[3] A. 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] A. W. Biermann and R. Krishnaswamy. Constructing programs from example computations. *IEEE Trans. on Software Engineering*, SE-2:141–153, 1976.

[5] K. Bogdanov and N. Walkinshaw. Computing the Structural Difference between State-Based Models. In *16th IEEE Working Conference on Reverse Engineering (WCRE)*, 2009.

[6] D. Combe, C. de la Higuera, and J. Janodet. Zulu: An interactive learning competition, 2009.

[7] J. Cook and A. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions*

[6]http://stamina.chefbe.net/

on Software Engineering and Methodology, 7(3):215–249, 1998.

[8] 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.

[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 *Proceedings of the International Colloqium on Grammatical Inference and Applications (ICGI'94)*, volume 862 of *LNAI*, pages 25–37. Springer Verlag, 1994.

[11] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation, Third Edition*. Addison-Wesley, 2007.

[12] J. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM: Journal of the ACM*, 46, 1999.

[13] 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)*, volume 1433, pages 1–12, 1998.

[14] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. 1(1), 2007.

[15] D. Lo and S. Khoo. SMArTIC: towards building an accurate, robust and scalable specification miner. In *SIGSOFT FSE*, pages 265–275, 2006.

[16] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 501–510, New York, NY, USA, 2008. ACM.

[17] S. Lucas. Gecco 2004 noisy dfa results, 2005. http://cswww.essex.ac.uk/staff/sml/gecco/results/NoisyDFA/NoidyDFAResults.html.

[18] J. Oncina and P. Garcia. Inferring Regular Languages in Polynomial Update Time. In *Pattern Recognition and Image Analysis*, volume 1, pages 49–61. 1992.

[19] S. Reiss and M. Renieris. Encoding program executions. In *ICSE*, pages 221–230. IEEE Computer Society, 2001.

[20] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage*, 45(4):427–437, 2009.

[21] B. Starkie, F. Coste, and M. van Zaanen. The omphalos context-free grammar learning competition. In G. Paliouras and Y. Sakakibara, editors, *Grammatical Inference: Algorithms and Applications; 7th International Colloquium, ICGI 2004*, volume 3264 of *LNCS/LNAI*, pages 16–27. Springer, 2004.

[22] B. Starkie, F. Coste, and M. van Zaanen. Progressing the state-of-the-art in grammatical inference by competition: The omphalos context-free language learning competition. *AI Communications*, 18(2):93–115, 2005.

[23] B. Starkie, M. van Zaanen, and D. Estival. The tenjinno machine translation competition. In *Grammatical Inference: Algorithms and Applications, 8th International Colloquium, ICGI 2006, Tokyo, Japan, September 20-22, 2006, Proceedings*, volume 4201 of *Lecture Notes in Artificial Intelligence*, pages 214–226, Berlin, 2006. Springer.

[24] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[25] T. Trapane and E. Lattman, editors. *Proteins: Structure, Function, and Bioinformatics*, volume 69. Wiley, 2007. CASP7 Proceedings.

[26] M. van Zaanen and J. Geertzen. Problems with evaluation of unsupervised empirical grammatical inference systems. In *International Colloquium on Grammatical Inference: Algorithms and Applications, (ICGI)*, volume 5278 of *Lecture Notes in Computer Science*, pages 301–303. Springer, 2008.

[27] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin. Reverse Engineering State Machines by Interactive Grammar Inference. In *14th IEEE International Working Conference on Reverse Engineering (WCRE)*, 2007.

[28] N. Walkinshaw, K. Bogdanov, and K. Johnson. Evaluation and Comparison of Inferred Regular Grammars. In *Proceedings of the International Colloquium on Grammar Inference (ICGI)*, St. Malo, France, 2008.