

Fixing the State Budget: Approximation of Regular Languages with Small DFAs

Graeme Gange^{1,*}, Pierre Ganty^{2,**}, and Peter J. Stuckey¹

¹ Department of Computing and Information Systems, University of Melbourne,
3010 Australia

² IMDEA Software Institute, Madrid, Spain

Abstract. Strings are pervasive in programming, and arguably even more pervasive in web programming. A natural abstraction for reasoning about strings are finite-automata. They are a well-understood formalism, and operations on them are decidable and well-known. But in practice these operations either blow up in size or in cost of operations. Hence the attractive automata representations become impractical. In this paper we propose reasoning about strings using small automata, by restricting the number of states available. We show how we can construct small automata which over-approximate the language specified by a larger automata, using discrete optimization techniques, both complete approaches and incomplete approaches based on greedy search. Small automata provide a strong basis for reasoning about strings in programming, since operations on small automata do not blow up in cost.

1 Introduction

Strings are pervasive in programs, and arguably even more pervasive in web programming. They also arise as a natural representation of system configurations for multi-agent systems where agents are linearly ordered [3].

To reason about such systems or programs one has to manipulate possibly infinite sets of strings or languages. To achieve effective reasoning, a natural abstraction for languages is to consider the class of regular languages which were shown to be sufficiently expressive to verify non-trivial properties. Regular languages are well-studied and are supported by multiple description formalism including automata-based representations. The usual operations required for abstract reasoning such as Boolean operations and usual tests such as inclusion and equivalence can be implemented in polynomial time on deterministic finite-state

* This work was supported by the Australian Research Council through grants DE160100568 and LP140100437.

** Pierre Ganty has been supported by the Madrid Regional Government project S2013/ICE-2731, *N-Greens Software - Next-Generation Energy-Efficient Secure Software*, and the Spanish Ministry of Economy and Competitiveness project No. TIN2015-71819-P, *RISCO - Rigorous analysis of Sophisticated Concurrent and distributed systems*.

automata. This contrasts with non-deterministic finite-state automata where inclusion is PSPACE-complete. This has to be taken with a grain of salt since there exist languages whose specification by non-deterministic finite-state automata are logarithmically more succinct than their smallest deterministic automata counterpart [14]. In practice, however, deterministic finite-state automata often blow up in representation size impeding the success automata-based techniques.

In this paper we examine the use of deterministic finite-state automata of bounded size as a way to achieve scalability of automata-based techniques. By bounding the size we combine the benefit of a small representation with the polynomial runtime of the operations and tests on finite-state automata. We consider k -state deterministic finite-state automata as our representation for languages. This restriction avoids the blow up in size, the size of the whole automata is $k^{|\Sigma|}$, and avoids the blow up in cost of operations, each operation is at most $k^2|\Sigma|$.

We start by investigating basic questions and show that, in general, there is no “minimum” k -state DFA that includes a language specified by n -state DFA with $n > k$. Here “minimum” is defined using language inclusion. This result is expected since the intersection of two languages defined by k -state DFAs is in general not representable precisely as a k -state DFA. Therefore we identify criteria besides language inclusion to select between two k -state DFA approximation when language inclusion alone is inconclusive.

To evaluate the effectiveness of those criteria, we formalize the problem of computing a k -state DFA approximating a given a n -state DFA as a search problem. Modelling the problem as a search problem allows to flexibly express the criteria we identified as objective to minimize.

Our first model as a search problem restricts the search space to those k -state DFA resulting from merging state of the n -state DFA in accordance with a k -block partition of the n states. State partitioning is the basis of minimization algorithms for DFAs. Using partitions result in a straightforward encoding of k -state approximation because language inclusion follows immediately from partitioning.

In a second model, we formalize the search problem in full-generality by considering all k -state DFAs. Therefore, in this encoding, we encode the constraint that the language of the k -state DFA includes that of the n -state DFA.

As a consequence of our result on the absence of a least k -state DFA the above search problems have no unique solution. Therefore we rely on the identified criteria to narrow the set of solutions using an objective function.

Modelling the approximation of an automaton as a search problem allows us to find all possible approximations, or find optimal approximations, in practice these problems are challenging to solve. Hence it is worth considering a greedy approach to tackling these problems. We first introduce a greedy approach for finding quotient automata approximations. Then, motivated by the ease in which quotient automata collapse, we introduce a greedy algorithm that preserves more of the structure of the automata while building decreasing size approximations, based on the idea of tracking language dominance of states of the automata.

2 Preliminaries

Strings. We assume a finite alphabet of symbols Σ . A *string* w is either the empty string ε or of the form cw' where c is a symbol in Σ and w' a string. The *length* of a string w , denoted $|w|$, is the number of symbols appearing in the string. We use array notation to lookup the symbols appearing in a string. Suppose $|w| = l$ then $w[i], 1 \leq i \leq l$ is the i^{th} symbol appearing in the string. We assume the reader is familiar with regular expressions.

Finite-state automata. A finite-state automaton (or simply automaton) is a tuple $R = \langle Q, \Sigma, \delta, q_0, F \rangle$ where Σ is an *alphabet*; Q is a finite set of *states* including the *initial state* q_0 and a set F of *accepting states*; and $\delta \subseteq Q \times \Sigma \times Q$ is a set of *transitions*. The size of an automata R , $size(R)$ is defined as $|Q|$.

A *transition* in automaton R from state q to q' , written $q \rightarrow q'$, exists if there is $(q, c, q') \in \delta$ for some $c \in \Sigma$. A *computation* for string w of length l in an automaton R is a sequence of transitions $q_0 \rightarrow q_1 \rightarrow \dots \rightarrow q_l$ where $(q_i, w[i+1], q_{i+1}) \in \delta$. An *accepting computation* for w in R from state q_0 is a computation for w in the automaton R where $q_l \in F$. The *language* of automaton R from state $q \in Q$, $L(q, R)$, is the set of strings w which have an accepting computation from state q . The *language* of automaton R , $L(R) = L(q_0, R)$.

A state q is said to be *accessible* if there is a sequence of transitions from q_0 to q ; *co-accessible* if there is a sequence of transitions from q to some state $q' \in F$; and *useful* if q is both accessible and co-accessible. An automaton is *trim* if all its states are useful. An automaton is *deterministic* (DFA) if δ denotes a (partial) function from $Q \times \Sigma$ into Q . In that case, we sometimes use the notation $\delta(q, c)$ where $q \in Q, c \in \Sigma$ to refer to q' where $(q, c, q') \in \delta$, when it exists. An automaton is said to be a *t-DFA* if it is deterministic and trim.

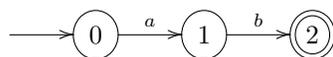
3 Automaton Approximation

In this paper we are interested in defining approximations of DFAs which are as precise as possible given a fixed budget on the number of states.

Definition 1. *Given two automata R and R' , we say that R' approximates R iff $L(R) \subseteq L(R')$. Given two k -state approximations R_1, R_2 of R we say R_1 dominates R_2 whenever $L(R_1) \subset L(R_2)$.*

When constructing an approximation of a DFA R we are interested in finding an approximation that is not dominated by any other approximation. Next we exhibit an example showing there might be more than one such approximation.

Example 1. Let A be the 3-state t-DFA $\langle \{0, 1, 2\}, \{a, b\}, \{(0, a, 1), (1, b, 2)\}, 0, \{2\} \rangle$ that accepts exactly $\{ab\}$.



Then for the three 2-state approximations shown in Figure 1: $L(B_1) = ab^*$, $L(B_2) = a^*b$ and $L(B_3) = (ab)^*$ and none dominates another. \square

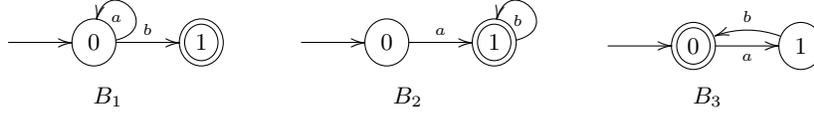


Fig. 1. Three 2 state approximations of the automata A .

While we are mainly interested in over-approximations of DFAs, we can define under-approximations as well using the complement. To compute an under-approximation of n -state DFA $R = (Q, \Sigma, \delta, q_0, F)$ such that δ is total, we complement R obtaining $\bar{R} = (Q, \Sigma, \delta, q_0, Q - F)$ and compute a k -state over-approximation of \bar{R} , $\bar{R}' = (Q', \Sigma, \delta', q_0', F')$ and then, assuming δ' is total,³ complement \bar{R}' to obtain $R' = (Q', \Sigma, \delta', q_0', Q' - F')$. R' is a k -state automaton under-approximating R by construction.

4 Approximations Using Equivalence Relations on States

DFA minimization relies on building an equivalence relation of the states of the DFA, or equivalently partitioning the states into equivalence classes. Given an n -state DFA R that minimizes to an equivalent k -state DFA R' , we find that R' dominates all other k -state approximations of R since $L(R') = L(R)$. This observation is the starting point of our study of approximations using state partitions.

4.1 Partitions and Quotient Automata

Consider a *partition* $P = \{b_1, \dots, b_n\}$ of Q into n non-empty, pairwise disjoint subsets covering Q called *blocks* and define the equivalence class of a state q , written $[q]_P$ as the (unique) block b_i such that $q \in b_i$. It is known that the set of partitions of a finite set forms a complete lattice. Thus we find that $(\text{Part}(Q), \preceq, \wedge, \vee, \{Q\}, \{\{q\} \mid q \in Q\})$ is a complete lattice where $\text{Part}(Q)$ is the set of partitions of Q ; $P_1 \preceq P_2$ iff for all blocks b_1 of P_1 there exist a block b_2 of P_2 such that $b_1 \subseteq b_2$; $P_1 \wedge P_2$ is the partition resulting from intersecting all pairs of blocks of P_1 and P_2 ⁴; $P_1 \vee P_2$ is the partition obtained by merging the blocks of P_1 which have a member in the same block of P_2 .

Given a t-DFA $R = \langle Q, \Sigma, \delta, q_0, F \rangle$ and a partition P of Q , then the *quotient automata* $R_{/P}$ is defined as the automaton $\langle P, \Sigma, \delta_P, [q_0]_P, F_P \rangle$ where $F_P = \{p \in P \mid p \cap F \neq \emptyset\}$, and $\delta_P = \{([q]_P, c, [q']_P) \mid (q, c, q') \in \delta\}$. Notice that $R_{/P}$ is not necessarily a t-DFA. The resulting automaton is a t-DFA for a subset of the partitions as we will see later.

A quotient automaton is always an approximation of the automaton it is defined from.

³ or a $k - 1$ -state over-approximation, on which we turn δ' into a total function.

⁴ Note that the empty set is not a block, hence it is not part of the resulting partition.

```

determinize-part( $P, \langle Q, \Sigma, \delta, q_0, F \rangle$ ):
   $T := \emptyset$ 
   $\delta' := \emptyset$ 
  for( $b_i \in P$ )
    for( $q \in b_i$ )
       $repr[q] := \min(b_i)$ 
  for( $(q, c, q') \in \delta$ )
    if( $(\text{FIND}(q), c, q'') \in \delta'$ )
       $T := T \cup \{(q', q'')\}$ 
    else  $\delta' := \delta' \cup \{(\text{FIND}(q), c, q')\}$ 
  while( $\exists(q_1, q_2) \in T$ )
     $T := T - \{(q_1, q_2)\}$ 
    UNION( $T, \delta', q_1, q_2$ )
   $Rs := \{q \mid q \in Q, \text{FIND}(q) = q\}$ 
  return  $\{q' \mid q' \in Q, \text{FIND}(q') = q\} \mid q \in Rs$ 

FIND( $x$ ):
  if( $repr[x] \neq x$ )
     $repr[x] := \text{FIND}(repr[x])$ 
  return  $repr[x]$ 

UNION( $T, \delta', x, y$ ):
   $rx := \text{FIND}(x)$ 
   $ry := \text{FIND}(y)$ 
  if( $rx = ry$ ) return
   $repr[ry] := rx$ 
  for( $(ry, c, q) \in \delta'$ )
     $\delta' := \delta' - \{(ry, c, q)\}$ 
  if ( $\exists(rx, c, q') \in \delta'$ )
     $T := T \cup \{(q, q')\}$ 

```

Fig. 2. Algorithm for computing $\det_R(P)$. It maintains a partition represented by a union-find data structure $repr$.

Theorem 1. Given t-DFA R and partition P of its states, $L(R) \subseteq L(R/P)$. \square

Example 2. Consider the t-DFA of Example 1, we have $B_1 = A/\{\{0,1\},\{2\}\}$, $B_2 = A/\{\{0\},\{1,2\}\}$ and $B_3 = A/\{\{0,2\},\{1\}\}$. \square

4.2 Determinizing Partitions

Note that the quotient automata of a t-DFA is not necessarily deterministic. A partition P for a t-DFA R is *deterministic* if for all $\{(q, c, q'), (q'', c, q''')\} \subseteq \delta$, $[q]_P = [q'']_P \Rightarrow [q']_P = [q''']_P$. Hence the resulting quotient automata R/P is also a t-DFA. Given a t-DFA R we define the *language quotients* of R as the automata R/P arising from all deterministic partitions P of R . Lemma 1 shows that the set of deterministic partitions forms a meet semi-lattice.

Lemma 1. Let P and P' be deterministic partitions for some t-DFA R . Then the partition $P \wedge P'$ is deterministic. \square

Lemma 2. Given a t-DFA R and partition P there is a least deterministic partition $\det_R(P)$ of R such that $P \preceq \det_R(P)$. \square

For any t-DFA R and partition P , we compute the least deterministic partition $\det_R(P)$ in $O(|\delta| + |P||\Sigma|)$ time. This algorithm is given in Figure 2.

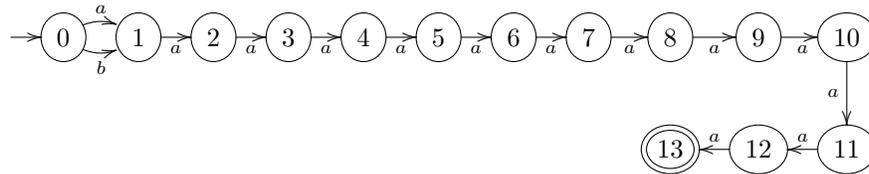
Lemma 3. Procedure $\text{determinize-part}(P, R)$ runs in $O(|\delta| + |P||\Sigma|)$ time. \square

Lemma 4. Let $P' = \text{determinize-part}(P, R)$. Then $P' = \det_R(P)$. \square

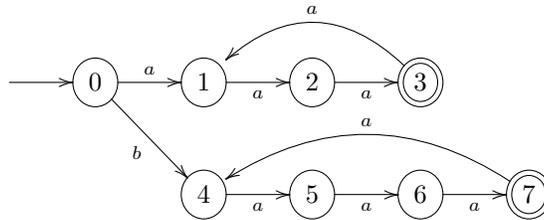
4.3 Incompleteness of Partition-Based Approaches

It would be convenient if the only k -state DFA approximations we need to consider were quotient automata, since there are many fewer quotient automata of an n -state t-DFA, than there are k -state DFA. Unfortunately, this is not the case.

Example 3. The following 14 state automaton A_1 for $(a|b)aaaaaaaaaaaa$



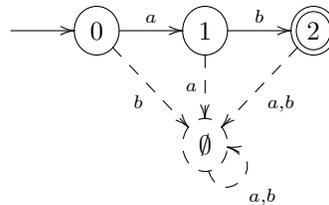
has this 8 state approximation A_2 such that $L(A_2) = aaa(aaa)^*|baaa(aaaa)^*$



There is no 8-state quotient of A_1 whose language is a subset of $L(A_2)$ since any quotient t-DFA must map state 1 to a single block and hence accepts a language of the form $(a|b)E$ for some regular expression E . \square

It is worth noting that when computing quotients of sparse automata, *trimming* is crucial. With a complete DFA, the additional error transitions force us to merge more states than we need to preserve determinism.

Example 4. Recall the automaton from Example 1, now as a complete DFA.



If we attempt to merge states 0 and 2 as before, the transitions $(0, a, 1)$ and $(2, a, \emptyset)$ force us to merge 1 with \emptyset . Then $(\emptyset, b, \emptyset)$ and $(1, b, 2)$ forces us to merge the two remaining partitions, yielding the single-state trivial automaton.

5 Approximations as a Search Problem

We define a discrete satisfaction problem that given an original n -state t-DFA $R = \langle Q, \Sigma, \delta, q_0, F \rangle$, finds a k -state DFA $R' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ which approximates it. In the next subsection we consider only quotient DFA, before generalizing this to arbitrary DFA in the following subsection.

5.1 Searching Quotient k -state DFAs

Quotient DFAs are an attractive class to consider for approximating arbitrary DFAs since they automatically satisfy the approximation condition, and they can be specified simply in terms of a partition.

Writing this problem as a combinatorial search problem is reasonably straightforward since we are merely deciding a partition. The model is defined by the principal decisions m_q which maps each state $q \in Q$ to a state in Q' where $|Q'| = k$ (which represent the states of the k -state DFA). The constraints are

$$m_q \in Q' \tag{1}$$

$$m_q \text{ is surjective} \tag{2}$$

$$q_0' = m_{q_0} \tag{3}$$

$$F' = \{m_q \mid q \in F\} \tag{4}$$

$$\delta' = \{(m_q, c, m_{q'}) \mid (q, c, q') \in \delta\} \quad \forall q \in Q, c \in \Sigma \tag{5}$$

$$\delta' \text{ is a partial function from } Q' \times \Sigma \text{ to } Q' \tag{6}$$

The size of the system of constraints $O(nk|\Sigma|)$.

Theorem 2 (Correctness). *Let t-DFA R and DFA R' satisfy equations (1)–(6) then $L(R) \subseteq L(R')$.* \square

Theorem 3 (Completeness). *Given t-DFA R and R' , if R' is a quotient t-DFA of R then there exist m satisfying equations (1)–(6).* \square

In practice we improve the model (1)–(6) by adding value symmetry breaking constraints to remove isomorphic k -states automata. We add a symmetry breaking constraint requiring that $\forall q \in Q, u \in \{2, \dots, k\}$ if $m_q = u$ then there exists $q' < q, m_{q'} = u - 1$. This enforces that we number the partitions P of R in the order of their least element. To be compatible with equation (3) we also require that the initial states q_0 and q_0' of both R and R' are the least numbered state.

5.2 Searching all k -state DFAs

Not all k -state DFAs are quotient automata. Furthermore, Example 3 shows there are k -state DFAs approximations which are dominated by no k -state quotient DFA. Hence we are also interested in non-quotient k -state DFA that are approximations.

To model the general approximation problem we reason about the synchronized product of the known n -state t-DFA, and the unknown k -state DFA. The principle decisions are δ' the transition relation for the k -state DFA and F' the set of final states. We ensure that each state reachable in the synchronized product, which represents a final state for the n -state t-DFA, is also a final state for the k -state DFA. The propositional decision variables $r_{q,q'}$ represent that the state (q, q') is reachable in the intersection DFA. The first four constraints ensure that any computation in the n -state t-DFA is reflected in the k -state DFA. The last constraint ensures that each reachable state in the synchronized product which is final for the n -state t-DFA is also final for the k -state DFA.

$$r_{q_0, q_0'} \quad (7)$$

$$r_{q, q'} \rightarrow \forall (q, c, q_2) \in \delta \exists (q', c, q_2') \in \delta' : r_{q_2, q_2'} \quad (8)$$

$$\delta' \quad \text{is a partial function from } Q' \times \Sigma \text{ to } Q' \quad (9)$$

$$r_{q, q'} \rightarrow q' \in F' \quad \forall q \in F, q' \in Q' \quad (10)$$

The size of this system of constraints is $O(nk|\Sigma|)$.

Theorem 4 (Correctness). *Let t-DFA R and DFA R' satisfy equations (7)–(10) then $L(R) \subseteq L(R')$. \square*

Theorem 5 (Completeness). *Given t-DFAs R and R' if R' is k -state t-DFA such that $L(R) \subseteq L(R')$ then there is a solution of equations (7)–(10). \square*

5.3 Complexity

We conjecture that the problem of finding a non-dominated k -state DFA of an n -state t-DFA is NP-hard, even when we restrict to quotient automata.

NONDOMPART(R, P): given an n -state t-DFA R and deterministic partition P of R decide whether there exists a deterministic partition P' of R where $|P'| \leq |P|$ such that $L(R_{/P'}) \subset L(R_{/P})$.

NONDOMAPPROX(R, R'): given an n -state t-DFA R and a k -state DFA R' where $L(R) \subseteq L(R')$ decide whether there exists a k -state DFA R'' such that $L(R) \subseteq L(R'') \subset L(R')$.

Conjecture 1. NONDOMPART and NONDOMAPPROX are NP-hard \square

Observe that both problems are in NP. For NONDOMPART, guess a partition P' of the n states of R with no more than $|P|$ blocks; check in polynomial time that P' is deterministic; if successful then build the DFAs $R_{/P}$ and $R_{/P'}$; check, in polynomial time, that $L(R_{/P'}) \subseteq L(R_{/P})$ and $L(R_{/P'}) \neq L(R_{/P})$. The argument to show NONDOMAPPROX belongs to NP is similar.

There are some closely related problem which are NP-complete. Gold [8] shows that deciding if there exists a k -state automaton that agrees with a set of examples and counterexamples is NP-complete.

5.4 Objectives

The models above that describe the problem of finding a k -state approximation of an n -state t-DFA, while correctly capturing this question are not that useful in practice. It is always possible to answer with a single-state machine whose initial state is accepting and has self arcs for all symbols in the alphabet.

Ideally what we desire are k -state t-DFA R which are non-dominated. Although this is decidable it seems hard to compute, and we conjecture it is NP-hard. Instead we will consider simpler objectives which are easier to compute. Hence we convert our problem to a discrete optimization problem.

Counting Prefixes The first thing we consider is counting the number of strings accepted up to some length. We can compute the number of strings accepted $a_{q',l}$ for each state $q' \in Q'$ and each length from $l \in \{0, \dots, m\}$ as follows

$$a_{q',0} = q' \in F' \quad (11)$$

$$a_{q',l+1} = \sum_{(q',c,q'') \in \delta'} a_{q'',l} \quad (12)$$

The size of this constraint system is $O(k^2|\Sigma|m)$. We can then minimize the expected number of strings of length between 0 and m accepted by the initial state of the k -state DFA $\sum_{l=0, \dots, m} a_{q_0,l}$

Using a result of Moore [11] that states that a pair of automata can be differentiated by a string of length less than the sum of their numbers of states we show the following lemma.

Lemma 5. *Suppose $m \geq 2k-1$ then a k state DFA R' minimizing $\sum_{l=0, \dots, m} a_{q_0,l}$ is dominated by no k -state DFA R'' approximating R . \square*

Markov-like measures If we assume that the strings of interest have a Poisson distribution in length we can model this as an expected probability p_c for each alphabet symbol x and a probability pe of reaching the end of the string where $pe + \sum_{c \in \Sigma} p_c = 1$. We can define the expected proportion of strings r_q accepted by state q as a system of simultaneous equations

$$r_{q'} = pe \times (q' \in F') + \sum_{(q',c,q'') \in \delta'} p_c r_{q''} \quad (13)$$

The size of this constraint system is $O(k^2|\Sigma|)$. We can then minimize the expected proportion of strings accepted by the automata as $r_{q_0'}$.

Other objectives are possible: using a finite corpus of counter-examples, or calculating the expected proportion of strings of some length that are accepted.

```

quot( $R, k$ ):
  let  $R = \langle Q, \Sigma, \delta, q_0, F \rangle$ 
   $P := \{\{q\} \mid q \in Q\}$ 
   $m := +\infty$ 
  for( $\{b_1, b_2\} \subseteq P, b_1 \neq b_2$ )
     $P_{12} := \text{determinize-part}(P - \{b_1, b_2\} \cup \{b_1 \cup b_2\}, R)$ 
     $M' := R_{/P_{12}}$ 
    if  $m > \text{measure}(M')$ 
       $m := \text{measure}(M')$ 
       $M := M'$ 
  if  $\text{size}(M) \leq k$  return  $M$ 
  return quot( $M, k$ )

```

Fig. 3. Greedy algorithm for building a k -state quotient t-DFA for t-DFA R .

6 Greedy Approaches

Independent of whether our conjecture of NP hardness holds, solving the constraint optimization problems defined in the previous section are challenging. Their solving behaviour appears to scale exponentially in k in practice.

Hence we consider incomplete approaches to find k -state approximations to an n -state t-DFA, which may not necessarily return a non-dominated approximation, nor minimize any objective.

The most obvious approach to producing a greedy approximation is by restricting consideration to quotient DFA of the original n -state t-DFA R , and merging partitions in these DFA until we reach a DFA with k or less states.

The greedy algorithm is shown in Figure 3. At each stage it considers each deterministic automaton $R_{/P_{12}}$ that arises from merging each pair of states in the original automaton R , and calculates a measure (objective value) for the automaton. It greedily selects the best resulting t-DFA. If this has no more than k states the process finishes, otherwise we repeat the merging process.

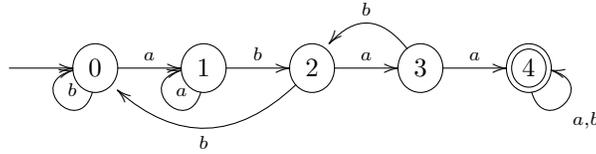
The result crucially depends on the measure used. Note that each of the objective measures defined in Section 5.4 is straightforward to calculate for a given fixed automaton $R_{/P_{12}}$.

Example 5. Consider the automaton A from Example 1. We will consider each of the quotient automata B_1, B_2 and B_3 of Figure 1 arising from partitions $\{\{0, 1\}, \{2\}\}$, $\{\{0\}, \{1, 2\}\}$ and $\{\{0, 2\}, \{1\}\}$ respectively. If we count the number of strings accepted of length up to 3, we find the measures are respectively 3, 3, and 2, and B_3 is preferable. If we use a Markov measure with $p_e = 1/3$ and $p_a = p_b = 1/3$ then the measures are respectively $1/6$, $1/6$ and $3/8$ and one of B_1 or B_2 is preferable. \square

7 Better State Merging

Quotient automata while easy to understand and construct often collapse quickly by partition determinization into small, and even single-state, automaton. Worse, some interesting classes of automata admit *no* non-trivial deterministic quotient.

Example 6. Consider the following automaton, recognizing the language $(a \cup b)^* abaa(a \cup b)^*$:



If we attempt to merge states 3 and 4, we find that the outgoing transitions on b conflict, and we are forced to add 2 to the partition. But as 2 transitions to 0 on b , we are again forced to merge 0 with $\{2, 3, 4\}$. But since 0 and $\{2, 3, 4\}$ disagree on a , we are finally forced to add 1 to the partition, obtaining the single-state universal automaton.

The same collapse occurs for any pair of states we choose to merge. However, in some sense it *should* be safe to merge state 3 into state 4, to produce automaton recognizing the shorter substring aba . \square

This suggests that language quotients describe too restrictive a form of transformation. Instead, consider some partition P such that $[q_1]_P = [q_2]_P$, $\{(q_1, c, q'_1), (q_2, c, q'_2)\} \subseteq \delta$, $[q'_1]_P \neq [q'_2]_P$ and $L(q'_1, R) \subseteq L(q'_2, R)$. If we replace the transition (q_1, c, q'_1) with (q_1, c, q'_2) , we resolve the non-determinism of δ_P and obtain an over-approximation of $L(R)$.

In order to use this we have to understand the inclusion relations between the states of the automaton we wish to approximate. To this end, we can use simulation preorders between states. Simulation preorders are computable in time polynomial in the size of the automaton. Moreover, if a simulation preorder \sqsubseteq is such that $q \sqsubseteq q'$ in R then $L(q, R) \subseteq L(q', R)$ holds [6].

We will determinize a t-DFA while shrinking the number of states from n to k by using an approximating version of the NFA to DFA translation. First we map the t-DFA with n -states R to a t-DFA with n -states RR whose state names are $QQ = \{\{q\} \mid q \in Q\}$. During the determinization we will construct new states which always represent sets of the original states Q . This mapping is only used to prevent creating states which already exists. We calculate the inclusion relation D for the t-DFA RR using, for instance, simulation preorders. We then apply the algorithm `determinize-tdfa` defined in Figure 4 to R and D choosing two states q_1 and q_2 in QQ to collapse.

The algorithm works by collapsing two states q_1 and q_2 into one. In the case that there is an inclusion relation between them this is easy, we simply eliminate the included state. Otherwise we create a new state q labelled with the union of original states of q_1 and q_2 , and replace all occurrences of q_1 and q_2 by q . This

```

determinize-tdfa( $R, D, q_1, q_2$ ):
  let  $R = \langle Q, \Sigma, \delta, q_0, F \rangle$ 
  if  $(q_1, q_2) \in D$ 
     $Q' := Q - \{q_1\}$ 
     $F' := F - \{q_1\}$ 
     $q_0' := (q_0 = q_1 ? q_2 : q_0)$ 
     $\delta' := \delta \cup \{(q', c, q_2) \mid (q', c, q_1) \in \delta\}$ 
    return  $\langle Q', \Sigma, (\delta' \cap (Q' \times \Sigma \times Q')), q_0', F' \rangle$ 
  elseif  $(q_2, q_1) \in D$ 
    return determinize-tdfa( $R, D, q_2, q_1$ ):
   $q := q_1 \cup q_2$  % new state
   $Q' := Q - \{q_1, q_2\} \cup \{q\}$ 
   $F' := F - \{q_1, q_2\} \cup \{q \mid q_1 \in F \vee q_2 \in F\}$ 
   $q_0' := (q_0 \in \{q_1, q_2\} ? q : q_0)$ 
   $D' := D \cup \{(q', q) \mid (q', q_1) \in D \vee (q', q_2) \in D\}$ 
   $\delta' := \delta \cup \{(q, c, q'') \mid (q', c, q'') \in \delta, q' \in \{q_1, q_2\}\} \cup \{(q', c, q) \mid (q', c, q'') \in \delta, q'' \in \{q_1, q_2\}\}$ 
   $\delta' := \delta' \cap (Q' \times \Sigma \times Q')$ 
   $R' := \langle Q', \Sigma, \delta', q_0', F' \rangle$ 
  while  $(\exists (q', c, q'_1) \in \delta', (q', c, q'_2) \in \delta', q'_1 \neq q'_2)$ 
     $(q'_{12}, \langle Q', \Sigma, \delta', q_0', F' \rangle, D') := \text{merge-state}(q'_1, q'_2, R', (D' \cap (Q' \times Q')))$ 
     $\delta' := \delta' - \{(q', c, q'_1), (q', c, q'_2)\} \cup \{(q', c, q'_{12})\}$ 
     $R' := \langle Q', \Sigma, \delta', q_0', F' \rangle$ 
  return  $\langle Q', \Sigma, (\delta' \cap (Q' \times \Sigma \times Q')), q_0', F' \rangle$ 

```

Fig. 4. Algorithm for computing a $n - 1$ -state DFA approximation of n -state t-DFA R given inclusion relation D which replaces q_1 and q_2 by their union.

may result in an NFA. We then continue finding non-deterministic transitions going to q'_1 and q'_2 and merging the states using `merge-state` which replaces one of these states by a state that over-approximate the union of their languages.

Lemma 6. *The algorithm `determinize-tdfa` terminates.* □

Lemma 7. *The algorithm `determinize-tdfa` returns a DFA R' with at most $n - 1$ states such that $L(R) \subseteq L(R')$.* □

We can adapt the greedy algorithm `quot` to use `determinize-tdfa` to construct the candidate automaton M' in place of `determinize-part` in an obvious manner. We call this `dom`.

There is a simpler variation of determinization using dominance. We add a universal accepting state a to the original automata (initially unconnected) and we only consider greedily merging other states q with a . This means that the first if condition in `determinize-tdfa` always holds, so determinization is simple. We call this greedy variant `univ`.

```

merge-state( $q_1, q_2, R, D$ ):
  let  $R = \langle Q, \Sigma, \delta, q_0, F \rangle$ 
  if  $q_1 \cup q_2 \in Q$ 
    return ( $q_1 \cup q_2, R, D$ )
  if  $(q_1, q_2) \in D$ 
    return ( $q_2, R, D$ )
  elseif  $(q_2, q_1) \in D$ 
    return ( $q_1, R, D$ )
   $q := q_1 \cup q_2$ 
   $r := ( |\{(q', c, q_1) \mid (q', c, q_1) \in \delta\}| > |\{(q', c, q_2) \mid (q', c, q_2) \in \delta\}| ? q_2 : q_1 )$ 
   $Q' := Q - \{r\} \cup \{q\}$ 
   $F' := F - \{r\} \cup \{q \mid q_1 \in F \vee q_2 \in F\}$ 
   $q_0' := (q_0 = r ? q : q_0)$ 
   $D' := D \cup \{(q', q) \mid (q', q_1) \in D \vee (q', q_2) \in D\} \cup \{(q_1, q), (q_2, q)\}$ 
   $\delta' := \delta \cup \{(q', c, q) \mid (q', c, r) \in \delta\} \cup \{(q, c, q') \mid (q'', c, q') \in \delta, q'' \in \{q_1, q_2\}\}$ 
  return ( $q, \langle Q', \Sigma, (\delta' \cap (Q' \times \Sigma \times Q')), q_0', F' \rangle, (D' \cap (Q' \times Q'))$ )

```

Fig. 5. Algorithm for computing an NFA with one of states q_1 and q_2 replaced with their union. It returns the name of the merged state, an NFA and a language inclusion relation for the states of the NFA.

8 Experiments

To evaluate the proposed approximation strategies, we generated a corpus of automata. For an instance `ty-1-m-d.xx`, we generated m random words W with length l over alphabet $\{1, \dots, d\}$. We then built an automaton of the given type `ty`: exact matching $\{w \in W\}$, prefix $\{w\Sigma^* \mid w \in W\}$, suffix $\{\Sigma^*w \mid w \in W\}$ or substring $\{\Sigma^*w\Sigma^* \mid w \in W\}$. We generated 10 instances for each combination of $l \in \{5, 10, 15, 20\}$, $m \in \{1, 3, 5, 10\}$, $d \in \{2, 3, 5, 10\}$, yielding 2560 automata with between 6 and 200 states.

For an automaton having n states, we constructed k -state approximations for $k \in \{n-2, \frac{n}{2}, \frac{n}{4}\}$ using the three greedy merging approaches: quotient determinization (`quot`), merge-into-universal (`univ`) and determinization with dominance (`dom`).

Results for the three greedy approaches are illustrated in Figure 6. For each initial automaton and target size k , we computed the number of strings of length $\leq 2k-1$ recognized by the approximation, and reported the average over the 10 instances for each combination of parameters.

It is interesting to note the differences in behaviour of `quot` and `univ`. For automata recognizing a single substring, `quot` cannot produce a non-trivial approximation – after merging any two states, determinization produces the universal automaton. But for these, `univ` constructs good (frequently minimal) approximations, recognizing a shorter prefix of the target substring. Conversely, `quot` produces much better approximations for exact string matching automata than

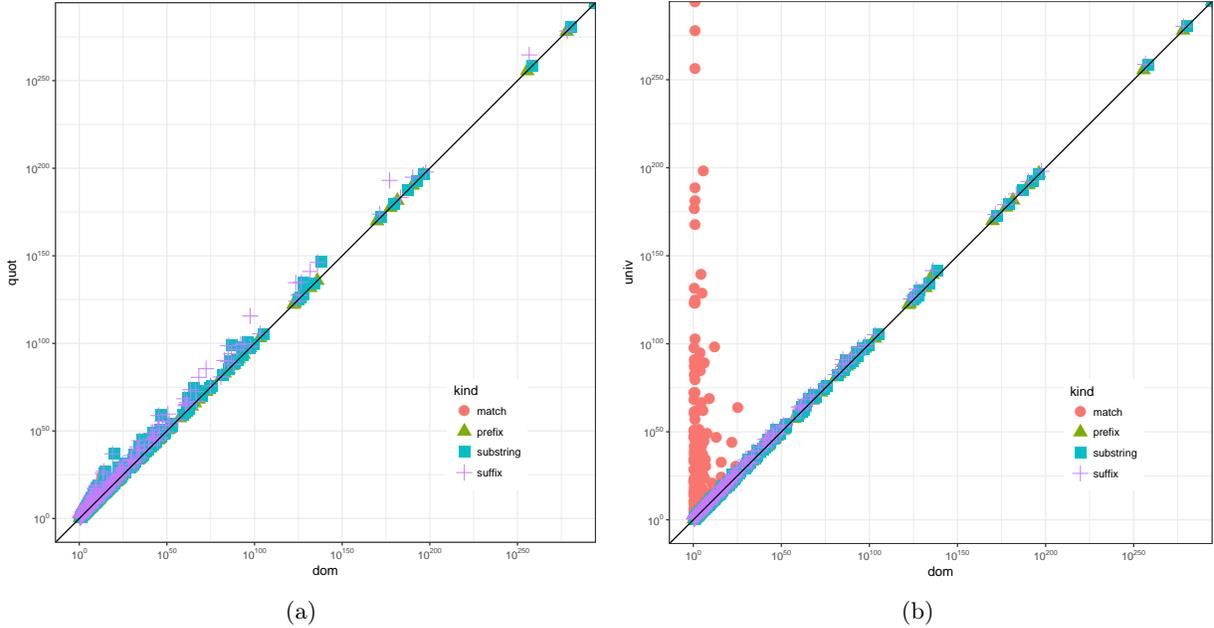


Fig. 6. For each parameter combination and target size k , the number of strings of length $\leq 2k-1$ recognized by the approximate automaton (averaged over 10 instances).

univ: univ produces automata matching all prefixes of the target string, whereas quot constructs long loops.

As hoped, the greater fidelity of **dom** allows us to achieve tighter approximations than **quot** or **univ**. In all but one instance, **dom** produced an automaton at least as tight, and frequently much tighter, than either **quot** or **univ**. A summary of runtimes is given in Figure 7.

In most cases approximations are constructed quickly. However for large automata, repeatedly evaluating all possible merges (each evaluation simulating the DFA up to $2k-1$ steps) becomes quite expensive. However, we expect by using previous results as lower bounds, we could avoid many of these evaluations entirely.

To test the accuracy of these greedy approximation strategies, we built MINIZINC [13] models for the optimal approximation obtainable in general (equations (7)–(10)), and under the **quot** and **univ** approximation strategies.⁵ We then computed optimal approximations of the smaller test instances using the

	med.	75%	max
quot	0.02	0.40	382.46
univ	< 0.01	0.04	29.32
dom	0.06	1.46	2819.01

Fig. 7. Runtime quartiles for the greedy approximation methods. The lower two quartiles are < 0.01 for all methods.

⁵ We do not include a model for **dom**, as the natural decision model is semantically equivalent to the general approximation model.

constraint programming solver CHUFFED [5]. CHUFFED was run with time and memory limits of 10 minutes and 2 Gb respectively. Figure 8 reports results for those automata where optimal approximations were found for all MiniZinc models.

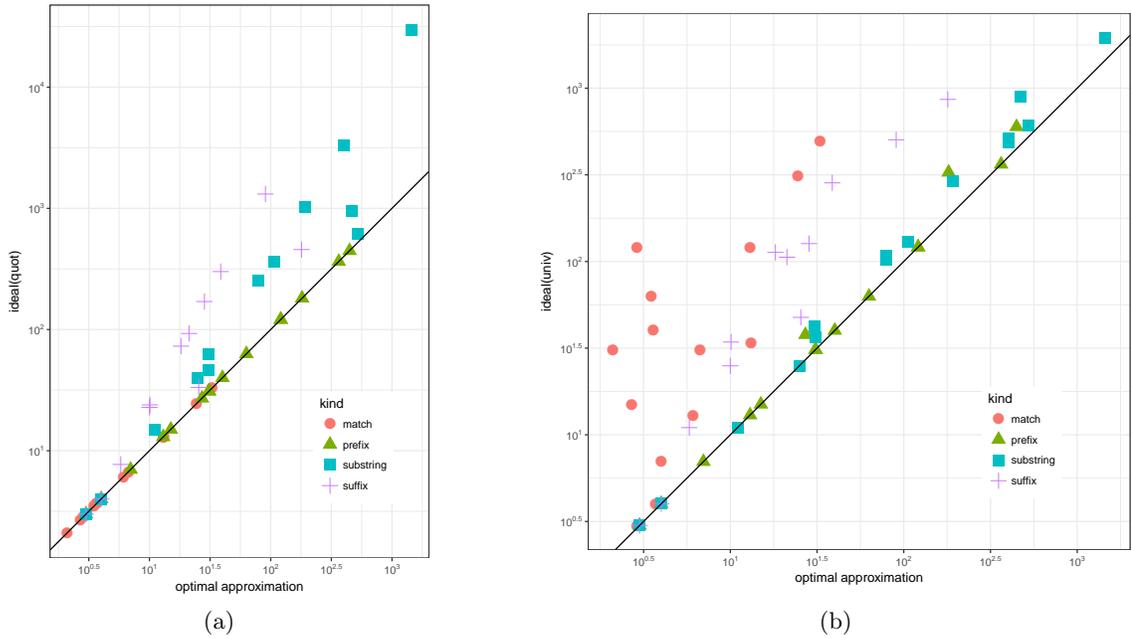


Fig. 8. Ideal approximations achievable under the (a) quot and (b) univ approximation strategies, compared with the best possible approximation.

The observed results match those for the greedy approaches: both quot and univ can represent optimal approximations for matching *single* prefixes, and produce poor approximations for suffix automata; but quot can produce optimal automata for exact matching, but univ produces much tighter (but typically not optimal) automata for substring matching.

9 Conclusion and Related work

Finding small approximating automata is a challenging problem, but the problem has a number of uses: in static analysis [7], in computer security [10], and elsewhere given the ubiquity of automata. In this paper we have formalized this problem as a search problem, and defined a number of complete and incomplete methods for finding correct approximations.

A related problem is that of computing minimal separating DFAs: Given two DFA defining disjoint languages, compute a DFA with as few states as possible

that includes one language and is disjoint from the other. Observe that the minimal separating DFA problem optimizes for size of the resulting DFA while we optimize for its precision with given size k . The minimal separating DFA problem has its origin in the learning of DFA from samples [1,8] where its formulation as a decision problem was shown to be NP-complete. More recently [9,4,12] the problem found applications in formal verification to discover invariants and in the context of assume guarantee reasoning. In that context, the work of Neider [12] is the most relevant to us since he defines a constraint-based approach to the minimal separating DFA problem.

On the other hand, the problem of finding minimal approximations using smaller automaton has been studied in several independent contexts [2,7,10]. Vijay D'silva [7] studied the problem in the context of static analysis where abstract states are given by languages of finite-state automaton. To ensure termination of fixpoint computations a widening operator has to be defined. D'silva lays down a principled approach to define widening operators for automata-based representation. Earlier Bouajjani et al. [2] faced identical termination problems in the context of abstract regular model-checking.

Both work put forward an approach based on state equivalences for various notions of equivalences. Equivalent states are then collapsed (merged or identified) yielding a smaller automaton whose language is a superset of original one. The equivalence relation is fixed a priori based on the application domain.

We differ in several aspects: we have a fixed budget on the number of states of the resulting automaton; and using our constraint-based approach we are searching a larger space of candidate automata, not necessarily automata resulting from merging states.

Finally, let us mention the work of Luchaup et al. [10] in the context of computer security where the use of approximations of finite-state automaton is motivated for performance reasons. Again they use an approach merging states with the goal of minimizing the error of classification.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. *Information and Computation* 75(2), 87–106 (1987)
2. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular (tree) model checking. *International Journal on Software Tools for Technology Transfer* 14(2), 167–191 (2011)
3. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: *Computer Aided Verification*, pp. 372–386. Springer (2004)
4. Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verification. In: *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 31–45. Springer (2009)
5. Chu, G.: Improving Combinatorial Optimization. Ph.D. thesis, Department of Computing and Information Systems, University of Melbourne (2011)
6. Dill, D.L., Hu, A.J., Wong-Toi, H.: Checking for language inclusion using simulation preorders. In: *Computer Aided Verification*, pp. 255–265. Springer (1992)

7. D'silva, V.: Widening for Automata. diploma thesis, Institut Für Informatik, Universität Zürich (2006)
8. Gold, E.M.: Complexity of automaton identification from given data. *Information and Control* 37(3), 302–320 (1978)
9. Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verification. *Formal Methods in System Design* 32(3), 285–301 (2008)
10. Luchaup, D., Carli, L.D., Jha, S., Bach, E.: Deep packet inspection with dfa-trees and parametrized language overapproximation. In: 2014 IEEE Conference on Computer Communications, INFOCOM. pp. 531–539. IEEE (2014)
11. Moore, E.F.: Gedanken-experiments on sequential machines. In: Shannon, C., McCarthy, J. (eds.) *Automata Studies*, pp. 129–153. Princeton University Press, Princeton, NJ (1956)
12. Neider, D.: Computing minimal separating DFAs and regular invariants using SAT and SMT solvers. In: *Automated Technology for Verification and Analysis*, pp. 354–369. Springer (2012)
13. Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In: *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*. LNCS, vol. 4741, pp. 529–543. Springer (2007)
14. Rabin, M.O., Scott, D.: Finite automata and their decision problem. *IBM Journal of Research and Development* 3, 114–125 (1959)