Electronic Notes in
Theoretical Informatics
and Computer Science

ENTICS
https://entics.episciences.org

Volume 1
Proceedings of
MFPS 2022

arXiv:2204.14153v4 [cs.FL] 5 Feb 2023

Guarded Kleene Algebra with Tests: Automata Learning

Stefan Zetzsche[a,1,3] Alexandra Silva[b,a,2] Matteo Sammartino[c,a]

[a] *University College London*
[b] *Cornell University*
[c] *Royal Holloway, University of London*

Abstract

Guarded Kleene Algebra with Tests (GKAT) is the fragment of Kleene Algebra with Tests (KAT) that arises by replacing the union and iteration operations of KAT with predicate-guarded variants. GKAT is more efficiently decidable than KAT and expressive enough to model simple imperative programs, making it attractive for applications to e.g. network verification. In this paper, we further explore GKAT's automata theory, and present GL^*, an algorithm for learning the GKAT automaton representation of a black-box, by observing its behaviour. A complexity analysis shows that it is more efficient to learn a representation of a GKAT program with GL^* than with Angluin's existing L^* algorithm. We implement GL^* and L^* in OCaml and compare their performances on example programs.

Keywords: Automata Learning, Kleene Algebra, Angluin, Coalgebra, Minimization, Moore Automata, Black-box, Model checking, Verification

1 Introduction

As hardware and software systems continue to grow in size and complexity, practical and scalable methods for verification tasks become increasingly important. Classical model checking approaches to verification require the existence of a rich model of the system of interest, able to express all its relevant behaviour. In reality such a model however is rarely available, for instance, when the system comes in the form of a black-box with no access to the source code, or the system is simply too complex for manual processing.

Automata learning, or regular inference, aims to automatically infer an automata model by observing the behaviour of the system. The incremental approach has been successfully applied to a wide range of verification tasks from finding bugs in network protocols [8], reverse engineering smartcard reader for internet banking [6], and industrial applications [14]. A comprehensive survey of the field can be found in [35]. The majority of modern learning algorithms is based on Angluin's L^* algorithm [3], which learns the unique minimal deterministic finite automaton (DFA) accepting a given regular language, or more generally, the unique minimal Moore automaton accepting a weighted language (Algorithm 1). In many

[1] The author has been supported by GCHQ via the VeTSS grant "Automated black-box verification of networking systems" (4207703/RFA 15845) and by the ERC via the Consolidator Grant AutoProbe 101002697.
[2] The author has been supported by the ERC via the Consolidator Grant AutoProbe 101002697 and by a Royal Society Wolfson Fellowship.
[3] Email: stefanzetzsche@gmail.com

Published February, 2023
Available online at https://doi.org/10.46298/entics.10505

© S. Zetzsche, A. Silva, M Sammartino
Creative Commons

Algorithm 1 Angluin's L^* algorithm for Moore automata with input A and output B

```
$S, E \leftarrow \{\varepsilon\}$
repeat
    while $T = (S, E, row : S \cup S \cdot A \to B^E)$ is not closed do
        find $t \in S \cdot A$ with $row(t) \neq row(s)$ for all $s \in S$
        $S \leftarrow S \cup \{t\}$
    end while
    construct and submit $m(T)$ to the teacher
    if the teacher replies *no* with a counterexample $z \in A^*$ then
        $E \leftarrow E \cup \mathtt{suf}(z)$
    end if
until the teacher replies *yes*
return $m(T)$
```

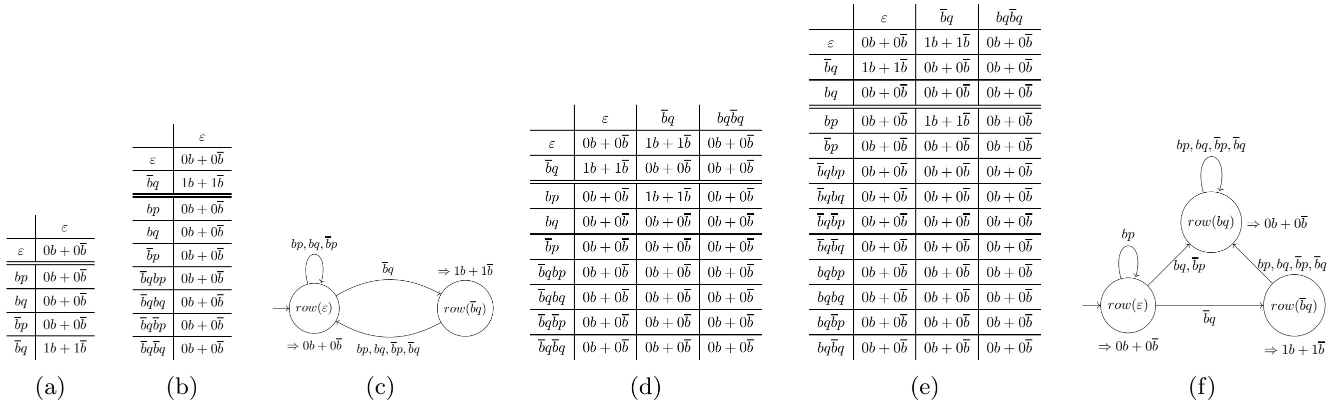
situations, however, targeting a DFA is not feasible, due to an explosion in the size of the state-space. Such cases instead require types of models specifically tailored for their domain-specific purposes.

For instance, modern networking systems can operate on very large data sets, making them very challenging to model. As a result, controlling, reasoning about, or extending networks can be surprisingly difficult. One approach to modernise the field that has recently gained popularity is *Software Defined Networking* (SDN) [10]. Modern SDN programming languages, notably *NetKAT* [2], allow operators to model their network and dynamically fine tune forwarding behaviour in response to events such as traffic shifts. Globally, NetKAT is based on *Kleene Algebra* (KA) [19], the sound and complete theory of regular expressions [18]. Locally, it incorporates *Boolean algebra*, the theory of predicates. Both logics have been unified in the well developed theory of *Kleene Algebra with Tests* (KAT) [20], which subsumes propositional Hoare logic and can be used to model standard imperative programming constructs. The automata theory for NetKAT has been introduced in [12].

Verifying properties about realistic networks reduces in NetKAT to deciding the behavioural equivalence of pairs of automata. Unfortunately, NetKAT's decision procedure is PSPACE-complete, mainly due its foundations in KAT. As a consequence, more efficiently decidable fragments of KAT have been considered. In [33] it was hinted that the *guarded fragment* of KAT is notably more efficiently decidable than the full language, while still remaining sufficiently expressive for networking purposes. The idea has been taken further in [32], which formally introduced *Guarded Kleene Algebra with Tests* (GKAT), a variation on KAT that arises by replacing the union and iteration operations from KAT with guarded variants. In contrast to KAT, the equational theory of GKAT is decidable in (almost) linear time. These properties make GKAT a promising candidate for the foundations of a SDN programming language that is more efficiently decidable than NetKAT.

In view of the potential applications of GKAT to the field of verification, this paper further investigates its automata theory. In detail, the paper makes the following contributions:

- For any GKAT automaton, we define a second automaton, which we call its minimization (Theorem 4.4). We show that in the class of normal GKAT automata, the minimization of an automaton is the unique size-minimal normal automaton accepting the same language (Theorem 4.12). We show that the minimization of a normal automaton is isomorphic to the automaton that arises by identifying semantically equivalent pairs among reachable states (Theorem 4.9), and that the minimizations of two language equivalent normal automata are isomorphic (Theorem 4.11). Finally, we show that minimizing a normal GKAT automaton preserves important invariants such as the nesting coequation (Theorem 4.10).
- We present GL^*, an active-learning algorithm (Algorithm 2) that incrementally infers a GKAT automaton from a black-box by querying an *oracle* (Section 5). We show that if the oracle is instantiated with the language accepted by a finite normal GKAT automaton, then the algorithm terminates with its minimization in finite time (Theorem 5.9).



	ε
ε	$0b+0\overline{b}$
bp	$0b+0\overline{b}$
bq	$0b+0\overline{b}$
$\overline{b}p$	$0b+0\overline{b}$
$\overline{b}q$	$1b+1\overline{b}$

(a)

	ε
ε	$0b+0\overline{b}$
$\overline{b}q$	$1b+1\overline{b}$
bp	$0b+0\overline{b}$
bq	$0b+0\overline{b}$
$\overline{b}p$	$0b+0\overline{b}$
$\overline{b}qbp$	$0b+0\overline{b}$
$\overline{b}qbq$	$0b+0\overline{b}$
$\overline{b}q\overline{b}p$	$0b+0\overline{b}$
$\overline{b}q\overline{b}q$	$0b+0\overline{b}$

(b)

$bp, bq, \overline{b}p$
$\overline{b}q$
$\Rightarrow 1b+1\overline{b}$
$row(\varepsilon)$
$row(\overline{b}q)$
$\Rightarrow 0b+0\overline{b}$
$bp, bq, \overline{b}p, \overline{b}q$
(c)

	ε	$\overline{b}q$	$bq\overline{b}q$
ε	$0b+0\overline{b}$	$1b+1\overline{b}$	$0b+0\overline{b}$
$\overline{b}q$	$1b+1\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
bp	$0b+0\overline{b}$	$1b+1\overline{b}$	$0b+0\overline{b}$
bq	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}p$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}qbp$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}qbq$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}q\overline{b}p$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}q\overline{b}q$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$

(d)

	ε	$\overline{b}q$	$bq\overline{b}q$
ε	$0b+0\overline{b}$	$1b+1\overline{b}$	$0b+0\overline{b}$
$\overline{b}q$	$1b+1\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
bq	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
bp	$0b+0\overline{b}$	$1b+1\overline{b}$	$0b+0\overline{b}$
$\overline{b}p$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}qbp$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}qbq$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}q\overline{b}p$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$\overline{b}q\overline{b}q$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$bqbp$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$bqbq$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$bq\overline{b}p$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$
$bq\overline{b}q$	$0b+0\overline{b}$	$0b+0\overline{b}$	$0b+0\overline{b}$

(e)

$bp, bq, \overline{b}p, \overline{b}q$
$row(bq)$
$\Rightarrow 0b+0\overline{b}$
bp
$bq, \overline{b}p$
$bp, bq, \overline{b}p, \overline{b}q$
$row(\varepsilon)$
$\overline{b}q$
$row(\overline{b}q)$
$\Rightarrow 0b+0\overline{b}$
$\Rightarrow 1b+1\overline{b}$
(f)

Fig. 1. An example run of Angluin's L^* algorithm for the target language $[\![(\texttt{while}\ b\ \texttt{do}\ p); q]\!]$.

- We show that the semantics of GKAT automata (2) can be reduced to the well-known semantics[4] of Moore automata. That is, there exists a language preserving embedding of GKAT automata into Moore automata (Theorem 6.1), which maps the minimization of a normal GKAT automaton to the language equivalent minimal Moore automaton (Theorem 6.2). In consequence, GKAT programs could thus, in principle, be also represented by Moore automata, instead of GKAT automata.
- We present a complexity analysis which shows that for GKAT programs it is more efficient to learn a GKAT automaton representation with GL^* than a Moore automaton representation with Angluin's L^* algorithm (Theorem 6.3). We implement GL^* and L^* in OCaml and compare their performances on example programs (Figure 6).

2 Overview of the approach

In this section, we give an overview of this paper through examples. We begin by presenting Algorithm 1, a slight variation of Angluin's L^* algorithm for finite Moore automata. We exemplify the algorithm by executing it for the language semantics of a simple GKAT program. We then propose a new algorithm, which, instead of a Moore automaton, infers a GKAT automaton.

2.1 L^* algorithm

Angluin's L^* algorithm learns the minimal DFA accepting a given regular language [3]. The algorithm has since been modified and generalised for a broad class of transition systems. The variation we present here step-wise infers the minimal Moore automaton accepting a generalised language $L : A^* \to B$ for a finite input set A and a finite output set B [27]. The algorithm assumes the existence of a *teacher* (or *oracle*), which can respond to two types of queries:

- **Membership queries**, consisting of a word $w \in A^*$, to which the teacher returns the output $L(w) \in B$;
- **Equivalence queries**, consisting of a hypothesis Moore automaton H, to which the teacher responds *yes*, if H accepts L, and *no* otherwise, providing a counterexample $z \in A^*$ in the symmetric difference of L and the language accepted by H.

The algorithm incrementally builds an *observation table*, which contains partial information about the language L obtained by performing membership queries. A table consists of two parts: a top part, with rows indexed by a finite set $S \subseteq A^*$; and a bottom-part, with rows ranging over $S \cdot A$. Columns are indexed by a finite set $E \subseteq A^*$. For any $t \in S \cup S \cdot A$ and $e \in E$, the entry at row t and column e, denoted

[4] In the language of Coalgebra, the semantics is given by the final coalgebra homomorphism for the functor defined by $FX = X^A \times B$, where $A = \mathrm{At} \cdot \Sigma = \{\alpha \cdot p \mid \alpha \in \mathrm{At},\ p \in \Sigma\}$ and $B = 2^{\mathrm{At}}$, for finite sets Σ and At. The carrier of the final coalgebra for F is $\mathcal{P}((\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At})$, the set of *guarded string languages*; the semantics of GKAT automata is given by the subclass of *deterministic* guarded string languages.

by $row(t)(e)$, is given by the output $L(te) \in B$. Note that the sets S and $S \cdot A$ can intersect. In such a case, elements in the intersection are only shown in the top part. Formally, we refer to a table as a tuple $T = (S, E, row)$, leaving the language L implicit.

Given a table T, one can construct a Moore automaton $m(T) = (X, \delta, \varepsilon, x)$, where $X = \{row(s) \mid s \in S\}$ is a finite set of states; the transition function $\delta : X \to X^A$ is given by $\delta(row(s), a) = row(sa)$; the output function $\varepsilon : X \to B$ satisfies $\varepsilon(row(s)) = row(s)(\varepsilon)$ (we abuse notation by writing ε both for the empty string and for the output function); and $x = row(\varepsilon)$ is the initial state. For $m(T)$ to be well-defined, the table T has to satisfy $\varepsilon \in S$ and $\varepsilon \in E$, and two properties called closedness and consistency. An observation table is *closed* if for all $t \in S \cdot A$ there exists an $s \in S$ such that $row(t) = row(s)$. An observation table is *consistent*, if whenever $s, s' \in S$ satisfy $row(s) = row(s')$, then $row(sa) = row(s'a)$ for all $a \in A$. A table is consistent in particular if the function row is injective.

The algorithm incrementally updates the table to satisfy those properties. If a well-defined hypothesis $m(T)$ can be constructed, the algorithm poses an equivalence query to the teacher, and either terminates, or refines the hypothesis with a counterexample $z \in A^*$. Since we respond to a negative equivalence query by adding the suffixes[5] of a counterexample to the set E (opposed to adding the prefixes of a counterexample to the set S), rows will always be distinct, rendering consistency trivial[6]. At all times, the set S is prefix-closed and the set E is suffix-closed[7].

2.1.1 *Example of execution*

We now execute Angluin's L^* (Algorithm 1) for the target language

$$L = [\![(\texttt{while}\ b\ \texttt{do}\ p); q]\!] = \{\overline{b}qb, \overline{b}q\overline{b}, bp\overline{b}qb, bp\overline{b}q\overline{b}, ...\} \subseteq (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}, \tag{1}$$

where $\mathrm{At} = \{b, \overline{b}\}$ is a finite set of *atoms* and $\Sigma = \{p, q\}$ is a finite set of *actions*. The language L represents the semantics of a program that performs the action p while b is true, and otherwise continues with q. It can be viewed as a generalised language $\widehat{L}$ with input $A = (\mathrm{At} \cdot \Sigma)$ and output $B = 2^{\mathrm{At}}$ via currying. We denote functions $f \in B$ as formal sums $\sum_{\alpha \in \mathrm{At}} f(\alpha)\alpha$. Each query to $\widehat{L}$ requires $|\mathrm{At}|$ many queries to L.

Initially, the sets S and E are set to the singleton $\{\varepsilon\}$. We build the observation table in Figure 1a. Since the row indexed by $\overline{b}q$ does not appear in the upper part, i.e. differs from the row indexed by ε, the table is not closed. To resolve the closedness defect we add $\overline{b}q$ to S. The observation table (Figure 1b) is now closed. We derive from it the hypothesis depicted in Figure 1c. Next, we pose an equivalence query, to which the oracle replies *no* and informs us that the word $z = bq\overline{b}q$ has been falsely classified. Indeed, given z, the language accepted by the hypothesis outputs $1b + 1\overline{b}$, whereas (1) produces $0b + 0\overline{b}$. To respond to the counterexample z, we add its suffixes to E. In this case, there are only the two suffixes $\overline{b}q$ and $bq\overline{b}q$. The next observation table (Figure 1d) again is not closed: the row indexed by e.g. bq does not equal any of the two upper rows indexed by ε and $\overline{b}q$. To resolve the closedness defect we add bq to S, and obtain the table in Figure 1e. The observation table is now closed. We derive from it the automaton in Figure 1f. Next, we pose an equivalence query, to which the oracle replies *yes*.

2.2 GL^* *algorithm*

In this section, we propose a new algorithm (Algorithm 2) for learning GKAT program representations, which we call GL^*. The new algorithm modifies Algorithm 1 by addressing a number of observations.

First, we note that the Moore automaton in Figure 1f admits multiple transitions to $row(bq)$, a *sink-state*, which does not accept any words. Second, we observe that languages induced by GKAT programs are *deterministic*[8]. Such languages are naturally represented by GKAT automata, which keep some

[5] The set $\texttt{suf}(z)$ of suffixes for $z \in A^*$ is defined by $\texttt{suf}(\varepsilon) = \{\varepsilon\}$ and $\texttt{suf}(aw) = \{aw\} \cup \texttt{suf}(w)$.

[6] This variation of L^* has been introduced by Maler and Pnueli [25].

[7] A set $X \subseteq A^*$ is called *suffix-closed*, if $\texttt{suf}(z) \subseteq X$ for all $z \in X$.

[8] Deterministic in the sense that, whenever two strings agree on the first n atoms, then they agree on their first n actions (or lack thereof).

Algorithm 2 The GL^* algorithm for GKAT automata

```
$S \leftarrow \{\varepsilon\}, E \leftarrow \mathrm{At}$
repeat
    while $T = (S, E, row : S \cup S \cdot (\mathrm{At} \cdot \Sigma) \to 2^E)$ is not closed do
        find $t \in S \cdot (\mathrm{At} \cdot \Sigma)$ with $row(t)(e) = 1$ for some $e \in E$, but $row(t) \neq row(s)$ for all $s \in S$
        $S \leftarrow S \cup \{t\}$
    end while
    construct and submit $m(T)$ to the teacher
    if the teacher replies no with a counterexample $z \in (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$ then
        $E \leftarrow E \cup \mathtt{suf}(z)$
    end if
until the teacher replies yes
return $m(T)$
```

transitions implicit. Third, in some cases[9] the deterministic nature of the target language allows us to fill-in parts of the observation table without performing any membership queries. Fourth, the cells of the observation table are labelled by functions, each of which requires two membership queries to (1); as a consequence, table extensions require an unfeasible amount of queries.

As before, we assume two finite sets, At and Σ, and a deterministic language $L \subseteq (\mathrm{At}\cdot\Sigma)^*\cdot\mathrm{At}$. The oracle of GL^* can answer two types of queries: membership queries consist of a word $w \in (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$, to which the oracle returns the output $L(w) \in 2$; equivalence queries consist of a hypothesis GKAT automaton H, to which the oracle responds *yes*, if H accepts L, and *no* otherwise, providing a counterexample $z \in (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$ in the symmetric difference of L and the language accepted by H.

An observation table in GL^* consists of two parts: a top part, with rows indexed by a finite set $S \subseteq (\mathrm{At} \cdot \Sigma)^*$; and a bottom-part, with rows ranging over $S \cdot \mathrm{At} \cdot \Sigma$. Columns range over a finite set $E \subseteq (\mathrm{At}\cdot\Sigma)^*\cdot\mathrm{At}$. The entry of the observation table at row t and column e, denoted by $row(t)(e)$, is given by $L(te) \in 2$. We refer to a table by $T = (S, E, row)$ and leave the deterministic language L implicit.

Given an observation table T, we construct a GKAT automaton $m(T) = (X, \delta, x)$, where $X = \{row(s) \mid s \in S\}$ is a finite set of states; $x = row(\varepsilon)$ is the initial state; and $\delta : X \to (2 + \Sigma \times X)^{\mathrm{At}}$ is the transition function which evaluates $\delta(row(s))(\alpha)$ to $(p, row(s\alpha p))$, if there exists an $e \in E$ with $row(s\alpha p)(e) = 1$; to 1, if $row(s)(\alpha) = 1$; and to 0, otherwise.

Most of the properties a table needs to satisfy such that the hypothesis $m(T)$ is well-defined are guaranteed by the construction of Algorithm 2, since L is deterministic. We only have to verify that the table is *closed*, that is, for all $t \in S\cdot\mathrm{At}\cdot\Sigma$ with $row(t)(e) = 1$ for some $e \in E$, there exists some $s \in S$ such that $row(t) = row(s)$. As in the case of L^*, the algorithm incrementally updates the table until closedness is guaranteed. It then constructs a well-defined hypothesis, and poses an equivalence query to the teacher. If the oracle replies *yes*, the algorithm terminates, and if the response is *no*, it adds the suffixes[10] of a counterexample $z \in (\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$ to E.

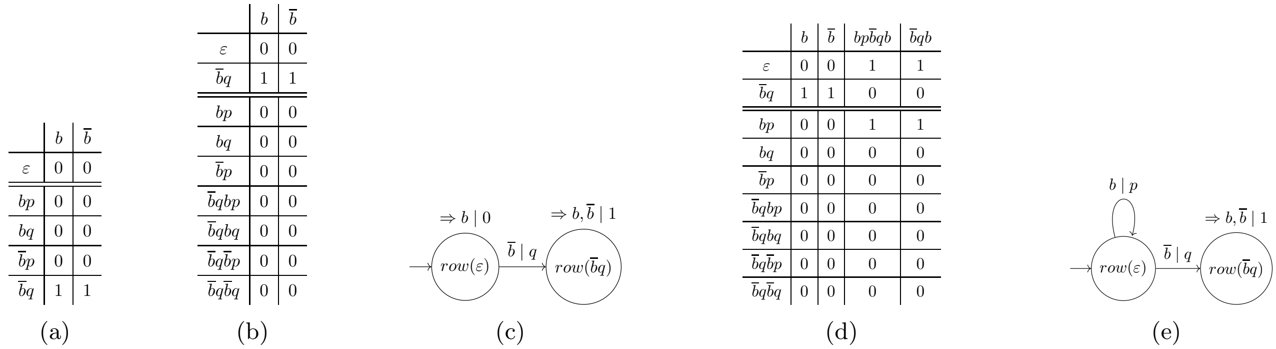
The differences between GL^* and L^* (instantiated for $A = \mathrm{At} \cdot \Sigma$ and $B = 2^{\mathrm{At}}$) are essentially a consequence of currying. In the former case, the set E contains elements of type $(\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$, and the table is filled with booleans in 2; in the latter case, the set E contains elements of type $(\mathrm{At} \cdot \Sigma)^*$, and the table is filled with functions $\mathrm{At} \to 2$. This, however, does not mean that GL^* is merely a shift in perspective: its new types induce independent definitions, and termination needs to be established with novel correctness proofs (Section 5). A thorough comparison with L^* is given in Section 6.

2.2.1 Example of execution

We now execute Algorithm 2 for the target language (1). Initially, $S = \{\varepsilon\}$ and $E = \mathrm{At}$. We build the observation table in Figure 2a. Since the bottom row indexed by $\bar{b}q$ contains a non-zero entry and differs

[9] For instance, the entries of the row indexed by bq in Figure 1d must all be zero, since the row indexed by bp admits a non-zero entry.

[10] The set $\mathtt{suf}(z)$ of suffixes for $z \in A^* \cdot B$ is defined by $\mathtt{suf}(wb) = \{vb \mid v \in \mathtt{suf}(w)\}$.



	b	$\overline{b}$
ε	0	0
bp	0	0
bq	0	0
$\overline{b}p$	0	0
$\overline{b}q$	1	1

(a)

	b	$\overline{b}$
ε	0	0
$\overline{b}q$	1	1
bp	0	0
bq	0	0
$\overline{b}p$	0	0
$\overline{b}qbp$	0	0
$\overline{b}qbq$	0	0
$\overline{b}q\overline{b}p$	0	0
$\overline{b}q\overline{b}q$	0	0

(b)

(c)

	b	$\overline{b}$	$bp\overline{b}qb$	$\overline{b}qb$
ε	0	0	1	1
$\overline{b}q$	1	1	0	0
bp	0	0	1	1
bq	0	0	0	0
$\overline{b}p$	0	0	0	0
$\overline{b}qbp$	0	0	0	0
$\overline{b}qbq$	0	0	0	0
$\overline{b}q\overline{b}p$	0	0	0	0
$\overline{b}q\overline{b}q$	0	0	0	0

(d)

(e)

Fig. 2. An example run of GL^* for the target language $[\![(\texttt{while}\ b\ \texttt{do}\ p); q]\!]$.

$$0 \equiv \texttt{false} \qquad 1 \equiv \texttt{true} \qquad t \equiv t \qquad b \cdot c \equiv b\ \texttt{and}\ c \qquad b + c \equiv b\ \texttt{or}\ c \qquad \overline{b} \equiv \texttt{not}\ b$$

$$p \equiv \texttt{do}\ p \qquad b \equiv \texttt{assert}\ b \qquad e \cdot f \equiv e; f \qquad e^{(b)} \equiv \texttt{while}\ b\ \texttt{do}\ e \qquad e +_b f \equiv \texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ f$$

Fig. 3. Identifying GKAT expressions with imperative programs.

from all upper rows (in this case, only the row indexed by ε), the table is not closed. We resolve the closedness defect by adding $\overline{b}q$ to S. The observation table (Figure 2b) is now closed. Note that the row indexed by $\overline{b}q$ indicates that the words $\overline{b}qb$ and $\overline{b}q\overline{b}$ are accepted. Since we know the target language is deterministic, the last four rows of the table can be filled with zeroes, without performing any membership queries. From Figure 2b we derive the hypothesis depicted in Figure 2c. Next, we pose an equivalence query, to which the oracle replies *no* and provides us with the counterexample $z = bp\overline{b}qb$, which is in the language (1), but not accepted by the hypothesis. We respond to the counterexample by adding its suffixes $bp\overline{b}qb$, $\overline{b}qb$ and b to E. The resulting observation table is depicted in Figure 2d. The table is closed, since the only non-zero bottom row is the one indexed by bp, which coincides with the upper row indexed by ε. Since the row indexed by bp has a non-zero entry, the row indexed by bq can automatically be filled with zeroes. We derive from Figure 2d the automaton in Figure 2e. Finally, we pose an equivalence query, to which the oracle replies *yes*.

3 Preliminaries

This section introduces the syntax and semantics of GKAT, an abstract imperative programming language with uninterpreted actions. For most parts, we follow the relevant bits of the original presentation in [32].

3.1 Syntax

The syntax of GKAT is inductively built from disjoint non-empty sets of *primitive tests*, T, and *actions*, Σ. In a first step, one generates from T a set of Boolean expressions, BExp. In a second step, the set is extended with Σ, to the full set of GKAT expressions, Exp:

$$\begin{aligned} b, c, d \in \mathrm{BExp} &::= 0 \mid 1 \mid t \in T \mid b \cdot c \mid b + c \mid \overline{b} \\ e, f, g \in \mathrm{Exp} &::= p \in \Sigma \mid b \in \mathrm{BExp} \mid e \cdot f \mid e +_b f \mid e^{(b)} \end{aligned}$$

By a slight abuse of notation, we will sometimes write ef for $e \cdot f$ and keep parenthesis implicit, e.g. $bc + d$ should be read as $(b \cdot c) + d$.

It is natural to view GKAT expressions as uninterpreted imperative programs. Under this view, one makes the identifications depicted in Figure 3.

Readers familiar with KAT will notice that the grammar for GKAT is similar to the one of KAT. It

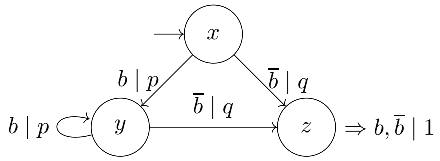


Fig. 4. The Thompson-automaton $\mathscr{X}_{p^{(b)}q}$ for $T = \{b\}$ and $\Sigma = \{p, q\}$.

differs in that GKAT replaces KAT's union (+) with the guarded union ($+_b$), and KAT's iteration (e^*) with the guarded iteration ($e^{(b)}$). GKAT's expressions can be encoded within KAT's grammar via the standard embedding that maps a conditional $e +_b f$ to $be + \overline{b}f$, and a while-loop $e^{(b)}$ to $(be)^*\overline{b}$.

3.2 Semantics: Language Model

In this section, we introduce the language semantics of GKAT, which assigns to a program the traces it could produce once executed. Intuitively, an execution trace is a string of the shape $\alpha_0 p_1 \alpha_1 ... p_n \alpha_n$. It can be thought of as a sequence of states α_i a system is in at point i in time, beginning with α_0 and ending in α_n, intertwined with actions p_i that transition from the state α_{i-1} to the state α_i.

More formally, let $\equiv_{\mathrm{BA}}$ denote the equivalence relation between Boolean expressions induced by the Boolean algebra axioms. The quotient $\mathrm{BExp}/_{\equiv_{\mathrm{BA}}}$, that is, the free Boolean algebra on generators T, admits a natural preorder $\leq$ defined by $b \leq c \Leftrightarrow b + c \equiv_{\mathrm{BA}} c$. The minimal nonzero elements with respect to this order are called *atoms*, the set of which is denoted by At. If $T = \{t_1, ..., t_n\}$ is finite, an atom $\alpha \in \mathrm{At}$ is of the form $\alpha = c_1 \cdot ... \cdot c_n$ with $c_i \in \{t_i, \overline{t_i}\}$.

A *guarded string* is an element of the set $\mathrm{GS} := \mathrm{At} \cdot (\Sigma \cdot \mathrm{At})^*$, or equivalently, $(\mathrm{At} \cdot \Sigma)^* \cdot \mathrm{At}$. The set of guarded strings without terminating atom is $\mathrm{GS}^- := (\mathrm{At} \cdot \Sigma)^*$.

A guarded string language $L \subseteq \mathrm{GS}$ is *deterministic* [32, Def. 2.2], if, whenever $\alpha_1 p_1 ... \alpha_{n-1} p_{n-1} \alpha_n v \in L$ and $\alpha_1 q_1 ... \alpha_{n-1} q_{n-1} \alpha_n w \in L$, then $p_i = q_i$ for all $1 \leq i \leq n-1$, and either $v = w = \varepsilon$, or $v = p_n v'$ and $w = q_n w'$ with $p_n = q_n$. The set of deterministic guarded string languages is denoted by $\mathscr{L}$.

Guarded strings can be partially composed via the *fusion product* defined by $v\alpha \diamond \beta w := v\alpha w$, if $\alpha = \beta$, and undefined otherwise. The partial product lifts to a total function on guarded languages by $L \diamond K := \{v \diamond w \mid v \in L, w \in K\}$. The n-th power of a guarded language is inductively defined by $L^0 := \mathrm{At}$ and $L^{n+1} := L^n \diamond L$. For $B \subseteq \mathrm{At}$ and $\overline{B} := \mathrm{At} \setminus B$, the guarded sum and the guarded iteration of languages are given by

$$L +_B K := (B \diamond L) \cup (\overline{B} \diamond K) \qquad L^{(B)} := \cup_{n \geq 0} (B \diamond L)^n \diamond \overline{B}.$$

The *language model* of GKAT is given by the semantic function $[\![-]\!] : \mathrm{Exp} \to \mathscr{P}(\mathrm{GS})$, which is inductively defined as follows:

$$[\![p]\!] := \{\alpha p \beta \mid \alpha, \beta \in \mathrm{At}\} \qquad [\![b]\!] := \{\alpha \in \mathrm{At} \mid \alpha \leq b\}$$
$$[\![e \cdot f]\!] := [\![e]\!] \diamond [\![f]\!] \qquad [\![e +_b f]\!] := [\![e]\!] +_{[\![b]\!]} [\![f]\!] \qquad [\![e^{(b)}]\!] := [\![e]\!]^{([\![b]\!])}.$$

Equivalently, the language semantics of GKAT can be constructed by post-composing the embedding of GKAT expressions into KAT expressions with the semantics of KAT.

The guarded string language $[\![e]\!]$ accepted by a GKAT program e is deterministic.

Example 3.1 Let the sets of primitive tests and actions be defined by $T := \{b\}$ and $\Sigma := \{p, q\}$, respectively. Then there exist only two atoms, $\mathrm{At} = \{b, \overline{b}\}$. The language model assigns to the program $p^{(b)}q \equiv (\texttt{while}\ b\ \texttt{do}\ p); q$ the guarded deterministic language (1).

3.3 Semantics: Automata Model

In this section, we introduce the automata model of GKAT, the central subject of this paper. As before, we assume two finite sets of tests T and actions Σ, the former of which induces a finite set of atoms, At.

Let G be the functor on the category of sets which is defined on objects by $GX = (2 + \Sigma \times X)^{\mathrm{At}}$, where $2 = \{0, 1\}$ is the two-element set, and on morphisms in the usual way. A *G-coalgebra* consists of a pair $\mathscr{X} = (X, \delta)$, where X is a set called *state-space* and $\delta : X \to GX$ is a function called *transition map*. A G-coalgebra *homomorphism* $f : (X, \delta^X) \to (Y, \delta^Y)$ is a function $f : X \to Y$ that commutes with the transition maps, $\delta^Y \circ f = Gf \circ \delta^X$. A *$G$-automaton* is a G-coalgebra $\mathscr{X}$ with a designated initial state $x \in X$. A homomorphism $f : (\mathscr{X}, x) \to (\mathscr{Y}, y)$ between G-automata is a homomorphism between the underlying G-coalgebras that maps initial state to initial state, $f(x) = y$.

For each state $x \in X$, given an input $\alpha \in \mathrm{At}$, a G-coalgebra either i) halts and accepts, that is, satisfies $\delta(x)(\alpha) = 1$; ii) halts and rejects, that is, satisfies $\delta(x)(\alpha) = 0$; or iii) produces an output p and moves to a new state y, that is, satisfies $\delta(x)(\alpha) = (p, y)$. Intuitively, for each state $x \in X$, a guarded string $\alpha_0 p_1 \alpha_1 ... p_n \alpha_n$ is accepted, if the G-coalgebra in state x produces the output $p_1 ... p_n$, halts and accepts. Formally, one defines a function $[\![-]\!] : X \to \mathscr{P}(\mathrm{GS})$ as follows:

$$\alpha \in [\![x]\!] :\Leftrightarrow \delta(x)(\alpha) = 1; \qquad \alpha p w \in [\![x]\!] :\Leftrightarrow \exists y \in X : \delta(x)(\alpha) = (p, y) \text{ and } w \in [\![y]\!]. \tag{2}$$

A G-coalgebra is *observable*, if the function $[\![-]\!]$ is injective.

A guarded string $w \in \mathrm{GS}$ is *accepted* by a state $x \in X$, if $w \in [\![x]\!]$. The language accepted by a G-automaton, $[\![\mathscr{X}]\!]$, is the language accepted by its initial state. Every language accepted by a G-automaton satisfies the determinacy property [32, Thm. 5.8]. Conversely, one can equip the set of deterministic languages with a G-coalgebra structure $(\mathscr{L}, \delta^{\mathscr{L}})$ defined by

$$\delta^{\mathscr{L}}(L)(\alpha) = \begin{cases} (p, (\alpha p)^{-1} L) & \text{if } (\alpha p)^{-1} L \neq \emptyset \\ 1 & \text{if } \alpha \in L \\ 0 & \text{otherwise} \end{cases},$$

where $(\alpha p)^{-1} L = \{ w \in \mathrm{GS} \mid \alpha p w \in L \}$. Since $[\![L]\!] = L$ for any $L \in \mathscr{L}$ [32, Thm. 5.8], every deterministic language can thus be recognized by a G-automaton with possibly infinitely many states.

A G-coalgebra (X, δ) is *normal*, if it only transitions to *live* states, that is, $\delta(x)(\alpha) = (p, y)$ implies $[\![y]\!] \neq \emptyset$, for all $x, y \in X$. For any G-automaton $\mathscr{X}$ one can construct a language equivalent normal G-automaton $\widehat{\mathscr{X}}$ [32, Lem. 5.6]. If $\mathscr{X}$ is normal, the function $[\![-]\!] : X \to \mathscr{P}(\mathrm{GS})$ is the unique coalgebra homomorphism $[\![-]\!] : (X, \delta) \to (\mathscr{L}, \delta^{\mathscr{L}})$ [32, Thm. 5.8].

Two states $x, y \in X$ of a normal coalgebra accept the same language, $[\![x]\!] = [\![y]\!]$, if and only if they are *bisimilar*, $x \simeq y$, that is, there exists a binary relation $R \subseteq X \times X$, such that, if xRy, then it holds:

- if $\delta(x)(\alpha) \in 2$, then $\delta(y)(\alpha) = \delta(x)(\alpha)$; and
- if $\delta(x)(\alpha) = (p, x')$, then $\delta(y)(\alpha) = (p, y')$ and $x' R y'$ for some $y' \in X$.

Bisimilarity is a symmetric relation and can be extended to two coalgebras by constructing a coalgebra that has the disjoint union of their state-spaces as state-space.

Using a construction that is reminiscent of Thompson's construction for regular expressions [34], it is possible to efficiently interpret a GKAT expression e as an automaton $\mathscr{X}_e$ that accepts the same language [32]. Alternatively, one can mirror [32] Kozen's syntactic form of Brzozowski's derivatives for KAT [22].

Example 3.2 The Thompson-automaton assigned to the expression $p^{(b)} q \equiv (\texttt{while } b \texttt{ do } p); q$ is depicted in Figure 4. It is normal, but not observable, since the states x and y are bisimilar, $x \simeq y$, thus accept the same language, $[\![x]\!] = [\![y]\!]$. Moreover, it is language equivalent to the expression by which it is generated, that is, it satisfies $[\![\mathscr{X}_{p^{(b)} q}]\!] = [\![p^{(b)} q]\!]$.

4 The minimal representation $m(\mathscr{X})$

The automaton $\mathscr{X}_e$ assigned to an expression e by the Thompson construction is not always the most efficient representation of the language $[\![e]\!]$. For instance, as seen in Theorem 3.2, the Thompson-automaton

$\mathscr{X}_{p^{(b)}q}$ in Figure 4 contains redundant structure, since its states x and y exhibit the same behaviour. In this section, we show that any G-automaton $\mathscr{X}$ admits an equivalent *minimal* representation, $m(\mathscr{X})$.

4.1 Reachability

We begin by formally defining what it means for a state of a G-automaton to be reachable, and show that restricting an automaton to its reachable states preserves important invariants.

Definition 4.1 Let (X, δ) be a G-coalgebra. We write $\to \subseteq X \times \mathrm{GS}^{-} \times X$ for the smallest relation satisfying:

$$\frac{}{x \xrightarrow{\varepsilon} x} \qquad \frac{\delta(x)(\alpha) = (p, y)}{x \xrightarrow{\alpha p} y} \qquad \frac{x \xrightarrow{\alpha_1 p_1 \ldots \alpha_{n-1} p_{n-1}} y \quad , \quad y \xrightarrow{\alpha_n p_n} z}{x \xrightarrow{\alpha_1 p_1 \ldots \alpha_n p_n} z}. \tag{3}$$

The states *reachable* from $x \in X$ are $r(x) := \{y \in X \mid \exists w \in \mathrm{GS}^{-} : x \xrightarrow{w} y\}$, and their *witnesses* are $R(x) := \{w \in \mathrm{GS}^{-} \mid \exists x_w \in X : x \xrightarrow{w} x_w\}$.

The following result shows that a state reached by a word is uniquely defined.

Lemma 4.2 *If $x \xrightarrow{w} x_w^1$ and $x \xrightarrow{w} x_w^2$, then $x_w^1 = x_w^2$.*

It is not hard to see that the subset $r(x)$ of reachable states is δ-invariant, i.e. if $y \in r(x)$ and $\delta(y)(\alpha) = (p, z)$, then $z \in r(x)$. We denote the well-defined sub-automaton one obtains by restricting to the states reachable from an initial state as $r(\mathscr{X})$, and call an automaton *reachable*, if $\mathscr{X} = r(\mathscr{X})$. Following [37, Def. 15], we call a normal, reachable, and observable automaton *minimal*.

The set $R(x)$ of words witnessing the reachability of states in $\mathscr{X} = (X, \delta, x)$ can be equipped with a G-automaton structure $R(\mathscr{X}) := (R(x), \partial, \varepsilon)$, where $\partial(w)(\alpha) = (p, w\alpha p)$, if $\delta(x_w)(\alpha) = (p, x_{w\alpha p})$ for some $x_{w\alpha p} \in X$, and $\partial(w)(\alpha) = \delta(x_w)(\alpha)$ otherwise. The automaton $r(\mathscr{X})$ can then be recovered as the image of the automata homomorphism $f : R(\mathscr{X}) \to \mathscr{X}$ defined by $f(w) = x_w$. In other words, there exists an epi-mono factorization $R(\mathscr{X}) \twoheadrightarrow r(\mathscr{X}) \hookrightarrow \mathscr{X}$.

We conclude with a list of important properties preserved by restricting an automaton to its reachable states. *Well-nestedness* and *coequations*, in particular, the *nesting coequation*, have been introduced in [32] and [31], respectively. We refer the reader to the original papers for formal definitions, and to Section 8 for a high-level comparison.

Proposition 4.3 *Let $\mathscr{X}$ be a G-automaton, then $r(\mathscr{X})$ is well-nested, normal, or satisfies the nesting coequation, whenever $\mathscr{X}$ does. Moreover, $r(\mathscr{X})$ accepts the same language as $\mathscr{X}$.*

4.2 Minimality

Recall that the state-space of the minimal DFA for a regular language consists of the equivalence classes of the so-called Myhill-Nerode equivalence relation [28].

Similarly, we define the state-space of the minimization of a GKAT automaton $\mathscr{X}$ as the equivalence classes of the equivalence relation $\equiv_{[\![\mathscr{X}]\!]}$ on GS^{-} defined for any guarded string language $L \subseteq \mathrm{GS}$ by:

$$v \equiv_L w :\Leftrightarrow \forall u \in \mathrm{GS} : vu \in L \text{ if(f) } wu \in L. \tag{4}$$

Let $v^{-1}L = \{u \in \mathrm{GS} \mid vu \in L\}$, then two words v, w are equivalent with respect to $\equiv_L$ if(f) their derivatives $v^{-1}L$ and $w^{-1}L$ coincide.

Definition 4.4 The *minimization* of a G-automaton $\mathscr{X} = (X, \delta, x)$ is $m(\mathscr{X}) := (\{w^{-1}[\![\mathscr{X}]\!] \mid w \in R(x)\}, \partial, [\![\mathscr{X}]\!])$ with

$$\partial(L)(\alpha) := \begin{cases} (p, (\alpha p)^{-1}L) & \text{if } (\alpha p)^{-1}L \neq \emptyset \\ 1 & \text{if } \alpha \in L \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

for $L \in \{w^{-1}[\![\mathscr{X}]\!] \mid w \in R(x)\}$.

A few remarks on the well-definedness of above definition are in order. The language accepted by a G-automaton is deterministic, and taking the derivative of a language preserves its deterministic nature. Thus only one of the three cases in (5) occurs. Since $\varepsilon \in R(x)$ and $\varepsilon^{-1}L = L$, the initial state of the minimization is well-defined. Transitioning to a new state is well-defined since $v^{-1}(w^{-1}L) = (wv)^{-1}L$.

It is not hard to see that on a high-level the minimization can be recovered as the image of the final automata homomorphism $[\![-]\!] : R(\mathscr{X}) \to \mathcal{L}$, which, as one verifies, satisfies $[\![w]\!]_{R(\mathscr{X})} = w^{-1}[\![\mathscr{X}]\!]$. In other words, there exists an epi-mono factorization $R(\mathscr{X}) \twoheadrightarrow m(\mathscr{X}) \hookrightarrow \mathcal{L}$.

4.2.1 Properties of $m(\mathscr{X})$

In this section we prove properties of $m(\mathscr{X})$, which one would expect to hold by a minimization construction. We begin by showing that minimizing a normal automaton results in a reachable acceptor.

Lemma 4.5 *Let $\mathscr{X}$ be a normal G-automaton with initial state $x \in X$. Then $[\![\mathscr{X}]\!] \xrightarrow{w} w^{-1}[\![\mathscr{X}]\!]$ in $m(\mathscr{X})$ for all $w \in R(x)$. In particular, $m(\mathscr{X})$ is reachable.*

The next result proves that minimizing an automaton preserves its language semantics.

Lemma 4.6 *Let $\mathscr{X}$ be a G-automaton, then $[\![L]\!] = L$ for all L in $m(\mathscr{X})$. In particular, $[\![m(\mathscr{X})]\!] = [\![\mathscr{X}]\!]$.*

An immediate consequence of above statement is that the states of the minimization can be distinguished by their observable behaviour, that is, different states accept different languages. Another implication of Theorem 4.6 is the normality of the minimization: all states are *live*.

Corollary 4.7 *Let $\mathscr{X}$ be a G-automaton, then $m(\mathscr{X})$ is normal and observable.*

Since $m(\mathscr{X})$ is normal, reachable, and observable, if $\mathscr{X}$ is normal, it is, by our definition, *minimal* (cf. [37, Def. 15]). Its size-minimality among normal automata language equivalent to $\mathscr{X}$ can be derived from the abstract definition, cf. Theorem 4.12.

4.2.2 Identifying $m(\mathscr{X})$

In this section, we identify the minimization of a normal G-automaton with an alternative, but equivalent, construction. In consequence, we are able to derive that the minimization of a normal automaton is size-minimal among language equivalent normal automata and preserves the nesting coequation. We begin by observing its universality in the following sense.

Proposition 4.8 *Let $\mathscr{X}$ and $\mathscr{Y}$ be normal G-automata with $[\![\mathscr{X}]\!] = [\![\mathscr{Y}]\!]$, and $y \in Y$ the initial state of $\mathscr{Y}$. Then $\pi : r(\mathscr{Y}) \to m(\mathscr{X})$ with $\pi(z) = w_z^{-1}[\![\mathscr{X}]\!]$, for $y \xrightarrow{w_z} z$ in $\mathscr{Y}$, is a (surjective) G-automata homomorphism, uniquely defined.*

The next result shows that the minimization of a normal G-automaton is isomorphic to the automaton that arises by identifying semantically equivalent pairs among reachable states.

Lemma 4.9 *Let $\mathscr{X}$ be a normal G-automaton with initial state $x \in X$ and $\pi : r(\mathscr{X}) \twoheadrightarrow m(\mathscr{X})$ as in Theorem 4.8, then $y \simeq z$ if(f) $\pi(y) = \pi(z)$ for all $y, z \in r(x)$. Consequently, $m(\mathscr{X})$ is isomorphic to $r(\mathscr{X})/\simeq$.*

$$\begin{tikzcd}
R(\mathscr{X}) \arrow[r, two heads] \arrow[dd, two heads] & r(\mathscr{X}) \arrow[d, hook] \arrow[ddl, dashed, two heads, "\pi"] \\
 & \mathscr{X} \arrow[d] \\
m(\mathscr{X}) \arrow[r, hook] & \mathcal{L}
\end{tikzcd}$$

(a) The morphism π as unique diagonal.

$$\begin{tikzcd}
e \arrow[d] & f \arrow[d] \\
\mathscr{X}_e \arrow[d] & \mathscr{X}_e \arrow[d] \\
\widehat{\mathscr{X}_e} \arrow[d] & \widehat{\mathscr{X}_f} \arrow[d] \\
m(\widehat{\mathscr{X}_e}) \arrow[r, leftrightarrow, "\cong"] & m(\widehat{\mathscr{X}_f})
\end{tikzcd}$$

(b) $[\![e]\!] = [\![f]\!]$ if(f) $m(\widehat{\mathscr{X}_e})$ and $m(\widehat{\mathscr{X}_f})$ are isomorphic.

Fig. 5. A high-level view of the notions introduced in Section 4.2.2.

On a high level, the automata homomorphism π can be recovered as the unique (surjective) diagonal making the diagram in Figure 5a commute.

In Theorem 4.3 it was noted that the reachable subautomaton $r(\mathscr{X})$ satisfies the nesting coequation, whenever $\mathscr{X}$ does. By Theorem 4.8 there exists an epimorphism $\pi : r(\mathscr{X}) \twoheadrightarrow m(\mathscr{X})$, if $\mathscr{X}$ is normal. Since coalgebras satisfying a coequation form a covariety, which is closed under homomorphic images [7,31], we thus can deduce the following result.

Corollary 4.10 *Let $\mathscr{X}$ be a normal G-automaton, then $m(\mathscr{X})$ satisfies the nesting coequation, whenever $\mathscr{X}$ does.*

We continue with the observation that two normal G-automata are language equivalent if and only if their minimizations are isomorphic. As depicted in Figure 5b, this implies that two expressions e and f are language equivalent if and only if the minimizations of their normalized Thompson automata are isomorphic. A similar idea occurs in Kozen's completeness proof for Kleene Algebra [19, Theorem 19].

Corollary 4.11 *Let $\mathscr{X}$ and $\mathscr{Y}$ be normal G-automata, then $[\![\mathscr{X}]\!] = [\![\mathscr{Y}]\!]$ if(f) $m(\mathscr{X}) \cong m(\mathscr{Y})$.*

We conclude with the size-minimality of the minimization of a normal automaton among language equivalent normal automata.

Corollary 4.12 *Let $\mathscr{X}$ and $\mathscr{Y}$ be normal G-automata with $[\![\mathscr{X}]\!] = [\![\mathscr{Y}]\!]$. Then $|m(\mathscr{X})| \leq |\mathscr{Y}|$, where $|m(\mathscr{X})| = |\mathscr{Y}|$ if(f) $m(\mathscr{X}) \cong \mathscr{Y}$.*

5 Learning $m(\mathscr{X})$

In this section we formally investigate the correctness of GL^* (Algorithm 2). Our main result is Theorem 5.9, which shows that if the oracle is instantiated with a deterministic language accepted by a finite normal G-automaton $\mathscr{X}$, then GL^* terminates with a hypothesis isomorphic to $m(\mathscr{X})$.

For calculations, it will be convenient to use the following definition of an observation table. One can show that if the oracle is instantiated with a finite normal G-automaton, then one has a well-defined observation table at every step.

Definition 5.1 An *observation table* $T = (S, E, row)$ consists of subsets $S \subseteq \mathrm{GS}^-$, $E \subseteq \mathrm{GS}$ and a function $row : S \cup S \cdot (\mathrm{At} \cdot \Sigma) \to 2^E$, such that:

- $\varepsilon \in S$ and $\mathrm{At} \subseteq E$
- $\alpha p e \in E$ implies $e \in E$ (suffix-closed)
- $s \alpha p \in S$ implies $s \in S$ (prefix-closed)
- $s \neq t$ implies $row(s) \neq row(t)$ for $s, t \in S$
- $\varepsilon \neq s \in S$ implies $row(s)(e) = 1$ for some $e \in E$
- $row(s \alpha p)(e) = row(s)(\alpha p e)$, if $\alpha p e \in E$

Not every table induces a well-defined G-automaton. To ensure correctness, we have to restrict ourselves to a subclass of tables that satisfies two important properties. We call an observation table *deterministic* if the guarded string language $row(s) \subseteq \mathrm{GS}$ is deterministic for all $s \in S$. An observation table is *closed*, if for all $t \in S \cdot (\mathrm{At} \cdot \Sigma)$ with $row(t)(e) = 1$ for some $e \in E$, there exists an $s \in S$ such that $row(s) = row(t)$.

Definition 5.2 Given a closed deterministic observation table $T = (S, E, row)$, let $m(T) := (\{row(s) \mid s \in S\}, \delta, row(\varepsilon))$ be the G-automaton with

$$\delta(L)(\alpha) = \begin{cases} (p, (\alpha p)^{-1}L) & \text{if } (\alpha p)^{-1}L \neq \emptyset \\ 1 & \text{if } \alpha \in L \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

where $L \in \{row(s) \mid s \in S\}$ and $(\alpha p)^{-1} row(s) = row(s\alpha p)$.

A few remarks on the well-definedness of above definition are in order. By Theorem 5.1 the upper-rows of an observation table are disjoint. Since T is deterministic, precisely one of the three cases in (6) occurs. If $(\alpha p)^{-1} row(s)$ is non-empty, there exists, because T is closed, some $t \in S$ with $(\alpha p)^{-1} row(s) = row(t)$. This shows that $m(T)$ is closed under transitions.

5.1 Properties of $m(T)$

In what follows, let T be a closed deterministic observation table, unless states otherwise. We will establish a few basic properties of $m(T)$. First, we observe its reachability, which is implied by a stronger statement.

Lemma 5.3 *It holds $row(s) \xrightarrow{t} row(st)$ in $m(T)$ for all $s \in S$ and $t \in \mathrm{GS}^-$, such that $st \in S$. In particular, $m(T)$ is reachable.*

We call a G-automaton $(\mathscr{Y}, y)$ *consistent with* T, if $S \subseteq R(y)$ and $[\![y_s]\!](e) = row(s)(e)$ for all $s \in S$, $e \in E$, and $y_s \in Y$ with $y \xrightarrow{s} y_s$. By Theorem 5.3, the automaton $m(T)$ is consistent with T if and only if $[\![row(s)]\!](e) = row(s)(e)$ for all $s \in S$ and $e \in E$. The consistency of $m(T)$ with T should not be confused with the consistency of T itself. Both terminologies appear frequently in the literature [3]. We show that $m(T)$ is not only consistent with T, but has in fact the fewest number of states among all automata consistent with T.

Lemma 5.4 *$m(T)$ is size-minimal among automata consistent with T.*

From the consistency of $m(T)$ with T it is straightforward to derive its normality and observability.

Lemma 5.5 *$m(T)$ is normal and observable.*

5.2 Relationship between $m(T)$ and $m(\mathscr{X})$

We will next deduce the correctness of GL^*, that is, its termination with an automaton isomorphic to $m(\mathscr{X})$, if the teacher is instantiated with the language accepted by a finite normal automaton $\mathscr{X}$.

In a first step we establish that any hypothesis admits an injective function from its state-space into the state-space of $m(\mathscr{X})$. The result below does not necessarily require the observation table to be deterministic or closed.

Lemma 5.6 *Let $T = (S, E, row)$ be an observation table with $row(t)(e) = [\![\mathscr{X}]\!](te)$ for all $t \in S \cup S \cdot (\mathrm{At} \cdot \Sigma)$, $e \in E$, and let $x \in X$ be the initial state of $\mathscr{X}$. Then $\pi : \{row(s) \mid s \in S\} \to \{w^{-1}[\![\mathscr{X}]\!] \mid w \in R(x)\}$, $row(s) \mapsto s^{-1}[\![\mathscr{X}]\!]$ is a well-defined injective function.*

If the algorithm terminates with a hypothesis $m(T)$, the latter is, by definition, language equivalent to $\mathscr{X}$, and thus to the minimization $m(\mathscr{X})$, by Theorem 4.6. The next result implies a stronger statement: in case of termination, the hypothesis $m(T)$ is *isomorphic* to $m(\mathscr{X})$, via the function π of Theorem 5.6.

Proposition 5.7 *Let $T = (S, E, row)$ be a closed deterministic observation table with $row(t)(e) = [\![\mathscr{X}]\!](te)$ for all $t \in S \cup S \cdot (\mathrm{At} \cdot \Sigma)$, $e \in E$. Let π be the injection of Theorem 5.6, and $\mathscr{X}$ normal. The following are equivalent:*

(i) *$\pi : m(T) \simeq m(\mathscr{X})$ is a G-automata isomorphism;*

(ii) *$[\![m(T)]\!] = [\![m(\mathscr{X})]\!]$.*

The main argument in the proof of Theorem 5.9 is the result below. It shows that if the oracle replies *no* to an equivalence query and provides us with a counterexample z, then the table extended with the suffixes of z can immediately be closed only if it is the first time such a situation occurs.

Proposition 5.8 *Let $T = (S, E, row)$ be a closed deterministic observation table with $row(t)(e) = [\![\mathscr{X}]\!](te)$ for all $t \in S \cup S \cdot (\mathrm{At} \cdot \Sigma)$, $e \in E$. Let $[\![m(T)]\!](z) \neq [\![\mathscr{X}]\!](z)$ for some $z \in \mathrm{GS}$, and $T' = (S, E \cup \mathtt{suf}(z), row')$ with $row'(t)(e) = [\![\mathscr{X}]\!](te)$. If T' is closed, then $row'(\varepsilon)(e) = 0$ for all $e \in E$, but $row'(\varepsilon)(z') = 1$ for some $z' \in \mathtt{suf}(z)$.*

In consequence, an infinite chain of negative equivalence queries and immediately closed extended tables is impossible. Since fixing a closedness defect increases the size of $m(T)$, which by Theorem 5.6 is bounded by the finite number of states in $m(\mathscr{X})$, we can deduce the correctness of Algorithm 2.

Theorem 5.9 *If Algorithm 2 is instantiated with the language accepted by a finite normal automaton $\mathscr{X}$, then it terminates with a hypothesis isomorphic to $m(\mathscr{X})$.*

6 Comparison with Moore automata

How are the minimal GKAT automaton (Figure 2e) and the minimal Moore automaton (Figure 1f) representing the language (1) related? Why should we learn the former, and not the latter? Are there optimizations for L^* that we could adapt for GL^*? Those are the questions this section seeks to answer.

6.1 Embedding of GKAT automata

Comparing the GKAT automaton in Figure 2e with the Moore automaton (with input $\mathrm{At} \cdot \Sigma$ and output 2^{At}, short *M-automaton*) in Figure 1f suggests that the latter can be recovered from the former by adding a sink-state with which halting transitions can be made explicit. The result below formalises this idea. The language semantics of Moore automata is defined as usual.

Lemma 6.1 *Given a G-automaton $\mathscr{X} = (X, \delta, x)$, let $f(\mathscr{X}) := (X + \{\star\}, \langle \partial, \varepsilon \rangle, x)$ be the M-automaton with*

$$\partial(x)(\alpha p) := \begin{cases} y & \text{if } x \in X,\ \delta(x)(\alpha) = (p, y) \\ \star & \text{otherwise} \end{cases} \qquad \varepsilon(x)(\alpha) := \begin{cases} 1 & \text{if } x \in X,\ \delta(x)(\alpha) = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Then $[\![x]\!]_{\mathscr{X}} = [\![x]\!]_{f(\mathscr{X})}$ for all $x \in X$, and $[\![\star]\!]_{f(\mathscr{X})} = \emptyset$. In particular, $[\![f(\mathscr{X})]\!]_{f(\mathscr{X})} = [\![\mathscr{X}]\!]_{\mathscr{X}}$.

As one would hope for, above construction maps, up to isomorphism, the minimal GKAT automaton $m(\mathscr{X})$ to the minimal Moore automaton accepting the same language as $\mathscr{X}$.

Corollary 6.2 *Let $\mathscr{X}$ be a normal G-automaton, then $f(m(\mathscr{X})) \cong m(f(\mathscr{X}))$ as M-automata.*

6.2 Complexity analysis

We now compare the worst-case complexities of L^* (Algorithm 1) and GL^* (Algorithm 2) for learning automata representations of GKAT programs e. We are mainly interested in a bound to the number of membership queries to $[\![e]\!]$. The example runs in Figure 1 and Figure 2 seem to indicate that with respect to this aspect, GL^* performs better than L^*. The result below confirms this intuition.

Proposition 6.3 *Algorithm 1 requires at most $O(a*(|\mathrm{At}|*b))$ many membership queries to $[\![e]\!]$ for learning a M-automaton representation of e, whereas Algorithm 2 requires at most $O(a*(|\mathrm{At}|+b))$ many membership queries to $[\![e]\!]$ for learning a G-automaton representation of e, for some*[11] *integers $a, b \in \mathbb{N}$.*

One can show that for all integers x, y greater than 2, the product $x * y$ is strictly greater than the sum $x + y$. Moreover, the difference between $x * y$ and $x + y$ increases with the sizes of x and y. The advantage of $\mathtt{GL}^*$ over $\mathtt{L}^*$ for learning deterministic guarded string languages in terms of membership queries thus increases with the size of the set At, which is exponential in the number of primitive tests, $\mathrm{At} \cong 2^T$. In applications to network verification, the number of tests, thus atoms, is typically quite large [2]. The difference between $\mathtt{GL}^*$ and $\mathtt{L}^*$ described in Theorem 6.3 is mainly due to a subtle play with the table indices, based on currying. It can be further increased by avoiding querying certain rows all together, taking into account the deterministic nature of the target language, as indicated in Section 2.2.1.

6.3 Optimized counterexamples

In this section we present an optimization of $\mathtt{GL}^*$ that is based on a subtle refinement of Theorem 5.8. We show that, while Algorithm 2 reacts to a negative equivalence query with counterexample $z \in \mathrm{GS}$ by adding columns for *all* suffixes in $\mathtt{suf}(z)$, it is in fact enough to add columns for a smaller subset of suffixes $\mathtt{suf}(z') \subseteq \mathtt{suf}(z)$, for some $z' \in \mathtt{suf}(z)$ of minimal length. Our approach is inspired by the optimized counterexample handling method of Rivest and Schapire for $\mathtt{L}^*$ [30].

Lemma 6.4 *Let $T = (S, E, row)$ be a closed deterministic observation table with $row(t)(e) = [\![\mathscr{X}]\!](te)$ for all $t \in S \cup S \cdot (\mathrm{At} \cdot \Sigma)$, $e \in E$. Let $[\![m(T)]\!](z) \neq [\![\mathscr{X}]\!](z)$ for some $z \in \mathrm{GS}$, and $z' := \min(A_z)$*[12]*. If $T' = (S, E \cup \mathtt{suf}(z'), row')$ with $row'(t)(e) = [\![\mathscr{X}]\!](te)$ is closed, then $row'(\varepsilon)(e) = 0$ for all $e \in E$, but $row'(\varepsilon)(z') = 1$.*

Let z_0 be the shortest suffix of z and z_i the suffix of z of length $|z_{i-1}| + 1$. The suffix $\min(A_z)$ can easily be computed in at most $|\mathtt{suf}(z)| - 1$ steps: verify whether $z_i \in A_z$, beginning with z_0; if positive, break and set $\min(A_z) := z_i$, otherwise loop with z_{i+1}.

For example, if T is the closed table in Figure 2b with the corresponding hypothesis $m(T)$ in Figure 2c and counterexample $z = bp\overline{b}qb$, then $z' = \min(A_z) = \overline{b}qb$, since $b \notin A_z$. Theorem 6.4 shows that, instead of adding columns for the two non-present suffixes $bp\overline{b}qb$ and $\overline{b}qb$ of z, it is sufficient to add only one column for the single non-present suffix $\overline{b}qb$ of z'. In this case, the counterexample z is relatively short, thus the number of avoided columns small; in general, however, the advantage can be more significant.

7 Implementation

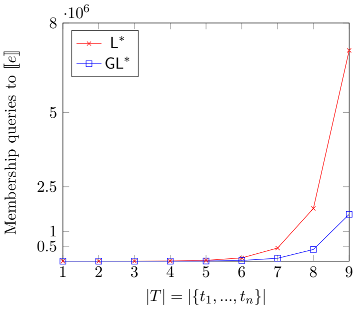
We have implemented both $\mathtt{GL}^*$ and $\mathtt{L}^*$ in OCaml; the code is available on GitHub[13]. The implementation allows one to compare, for any GKAT expression $e \in \mathrm{Exp}_{\Sigma,T}$, the number of membership queries to $[\![e]\!]$ required by $\mathtt{GL}^*$ for learning a G-automaton representation of e, with the number of membership queries to $[\![e]\!]$ required by $\mathtt{L}^*$ for learning a M-automaton representation of e. For each run, we output, for both algorithms, a trace of the involved hypotheses as tables in the `.csv` format and graphs in the `.dot` format, as well as an overview of the numbers of involved queries in the `.csv` format.

In Figure 6a we present the results for the expression $e = \mathtt{if}\ t_1\ \mathtt{then}\ \mathtt{do}\ p_1\ \mathtt{else}\ \mathtt{do}\ p_2$, the primitive actions $\Sigma = \{p_1, p_2, p_3\}$, and primitive tests $T = \{t_1, ..., t_n\}$ parametric in $n = 1, ..., 9$. We find that $\mathtt{GL}^*$ outperforms $\mathtt{L}^*$ for all choices of n. The difference in the number of membership queries increases with the size of n, as suggested by Theorem 6.3. For $n = 9$ the number of atoms is 2^9, resulting in an already relatively large number of queries for both algorithms. The picture is similar in Figure 6b, where we

[11] Let m be the maximum length of a counterexample and n the size of the minimal Moore automaton accepting $[\![e]\!]$, then $a = n * |\mathrm{At}| * |\Sigma|$ and $b = m * n$. As Figure 6 shows, $\mathtt{GL}^*$ can be more efficient than $\mathtt{L}^*$ even for small $|\mathrm{At}|$.

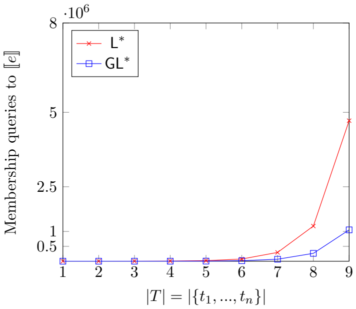
[12] $A_z := \{z' \in \mathtt{suf}(z) \mid z = v\alpha p z',\ row(\varepsilon) \xrightarrow{v} row(s_v),\ x \xrightarrow{s_v} x_{s_v},\ [\![row(s_v)]\!](\alpha p z') \neq [\![x_{s_v}]\!](\alpha p z')\}$

[13] https://github.com/zetzschest/gkat-automata-learning



$\lvert T \rvert$	GL^*	L^*
1	26	114
2	100	444
3	392	1.752
4	1.552	6.960
5	6.176	27.744
6	24.640	110.784
7	98.432	442.752
8	393.472	1.770.240
9	1.573.376	7.079.424

(a) $e =$ `if` t_1 `then do` p_1 `else do` p_2



$\lvert T \rvert$	GL^*	L^*
1	36	78
2	102	300
3	330	1.176
4	1.170	4.656
5	4.386	18.528
6	16.962	73.920
7	66.690	295.296
8	264.450	1.180.416
9	1.053.186	4.720.128

(b) $e = ($`while` t_1 `do` $p_1);$`do` p_2

Fig. 6. A comparison between GL^* and L^* with respect to membership queries.

choose the expression $e = (\texttt{while}\ t_1\ \texttt{do}\ p_1);\texttt{do}\ p_2$, the primitive actions $\Sigma = \{p_1, p_2\}$, and primitive tests $T = \{t_1, \ldots, t_n\}$ parametric in $n = 1, \ldots, 9$. Again, GL^* requires significantly less queries in all cases of n, and the difference increases with the size of n.

Our implementation generates an oracle for L^* from a GKAT expression e in the following way. First, we interpret e as a KAT expression $\iota(e)$ via the standard embedding of GKAT into KAT. Next, we generate from the latter a Moore automaton $\mathscr{X}_{\iota(e)}$ accepting $[\![e]\!]$, by using Kozen's syntactic Brzozowksi derivatives for KAT [22]. Finally, we answer an equivalence query from a Moore automaton $\mathscr{Y}$ by running a bisimulation between $\mathscr{X}_{\iota(e)}$ and $\mathscr{Y}$, similarly to [29, Fig. 1], and a membership query from $w\alpha \in \mathrm{GS}$ by returning the value of α at the output of the state in $\mathscr{X}_{\iota(e)}$ reached by w, that is, $[\![e]\!](w\alpha)$. A membership query from $w \in \mathrm{GS}^-$ is answered by querying $w\alpha \in \mathrm{GS}$ for all $\alpha \in \mathrm{At}$.

With the oracle for L^*, we can derive an oracle for GL^* as follows. Membership queries $w\alpha \in \mathrm{GS}$ are delegated and answered by the oracle for L^* as explained above. An equivalence query from a GKAT automaton $\mathscr{Y}$ is answered by posing an equivalence query to the oracle for L^* with the Moore automaton $f(\mathscr{Y})$ obtained via the embedding defined in Theorem 6.1. If the oracle for L^* replies with a counterexample $z \in \mathrm{GS}^-$, we extend z with an $\alpha \in \mathrm{At}$ such that $[\![\mathscr{Y}]\!](z\alpha) \neq [\![e]\!](z\alpha)$.

8 Related work

GKAT is a variation on KAT [23] that one obtains by restricting the union and iteration operations from KAT to guarded versions. While GKAT is less expressive than KAT, term equivalence is notably more efficiently decidable [32,23], making it a candidate for the foundations of network-programming [33,2,12]

GKAT automata appear in the literature already prior to [32], e.g. in the work of Kozen [24] under the name *strictly deterministic automata*. In the latter, Kozen states that GKAT automata correspond to a limited class of *automata with guarded strings (AGS)* [21], for which he gives determinization and minimization constructions. In a different paper [22] Kozen introduces a second definition of (deterministic) AGS as Moore automata, and states the difference to the definition of AGS in [21] is inessential.

Recently, a new perspective on the semantics and coalgebraic theory of GKAT has been given in terms of coequations [31,7]. Using the Thompson construction, it is possible to construct for every expression e a language equivalent automaton $\mathscr{X}_e$. In [24] it was shown that the inverse does generally not hold: there exists a GKAT automaton that is inequivalent to $\mathscr{X}_e$ for all expressions e. In consequence, [32] proposed a subclass of *well-nested* automata and showed that every finite well-nested automaton is bisimilar to $\mathscr{X}_e$ for some e. In [31] it was shown that well-nestedness is in fact too restrictive: there exists an automaton that is bisimilar to $\mathscr{X}_e$ for some e, but not well-nested. To capture the *full* class of automata exhibiting the behaviour of expressions, one has to extend the class of well-nested automata to the class of automata satisfying the *nesting coequation*, which forms a *covariety* [7].

Active automata learning is a technique used for deriving a model from a black-box by interacting with it via observations. The seminal algorithm L^*[3] learns deterministic finite automata, but since then has been extended to other classes of automata [4,1,26], including Moore automata. Typically, algorithms such as L^* are designed to output for a given language a unique minimal acceptor. Not all classes admit a canonical minimal acceptor, for instance, learning non-deterministic models is a challenge [9,5,39,38].

9 Discussion and future work

We have presented GL^*, an algorithm for learning the GKAT automaton representation of a black-box, by observing its behaviour via queries to an oracle. We have shown that for every normal GKAT automaton there exists a unique size-minimal normal automaton, accepting the same language: its minimization. We have identified the minimization with an alternative but equivalent construction, and derived its preservation of the nesting coequation. A central result showed that if the oracle in GL^* is instantiated with the language accepted by a finite normal automaton, then GL^* terminates with its minimization. A complexity analysis showed the advantage of GL^* over L^* for learning automata representations of GKAT programs in terms of membership queries. We discussed additional optimizations, and implemented GL^* and L^* in OCaml to compare their performances on example programs.

There are numerous directions in which the present work could be further explored. In Section 6.3 we introduced an optimization for GL^* which is inspired by Rivest and Schapire's counterexample handling method for L^* [30]. The *oberservation pack* algorithm for L^* [15] has successfully combined Rivest and Schapire's method with an efficient *discrimination tree* data structure [17]. The state-of-the-art *TTT*-algorithm [16] for L^* extends the former with discriminator finalization techniques. It thus is natural to ask whether for GL^* there exist similarly efficient data structures, potentially exploiting the deterministic nature of the languages accepted by GKAT automata.

While L^* has seen major improvements over the years and has inspired numerous variations for different types of transition systems, all approaches remain in common their focus on the *equivalence* of observations. The recently presented $\mathsf{L}^\sharp$ algorithm [36] takes a different perspective: it instead focuses on *apartness*, a constructive form of inequality. $\mathsf{L}^\sharp$ does not require data-structures such as observation tables or discrimination trees, instead operating directly on tree-shaped automata. It remains open whether a similar shift in perspective is feasible for GL^*.

There exist various domain-specific extensions of KAT (e.g. KAT+B! [13], NetKAT [2], ProbNetKAT [11]), and similar directions have been proposed for GKAT. In particular, it has been noted that GKAT is better fit for probabilistic domains than KAT, as it avoids mixing non-determinism with probabilities [33]. We expect that in the future, for such extensions of GKAT, there will be interest in developing the corresponding automata (learning) theories.

References

[1] Aarts, F. and F. Vaandrager, *Learning i/o automata*, in: *International Conference on Concurrency Theory*, Springer, 2010, pp. 71–85.
https://doi.org/10.1007/978-3-642-15375-4_6

[2] Anderson, C. J., N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger and D. Walker, *Netkat: Semantic foundations for networks*, ACM Sigplan Notices **49** (2014), pp. 113–126.
https://doi.org/10.1145/2578855.2535862

[3] Angluin, D., *Learning regular sets from queries and counterexamples*, Information and computation **75** (1987), pp. 87–106.
https://doi.org/10.1016/0890-5401(87)90052-6

[4] Angluin, D. and M. Csűrös, *Learning markov chains with variable memory length from noisy output*, in: *Proceedings of the tenth annual conference on Computational learning theory*, 1997, pp. 298–308.
https://doi.org/10.1145/267460.267517

[5] Bollig, B., P. Habermehl, C. Kern and M. Leucker, *Angluin-style learning of nfa*, in: *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
https://dl.acm.org/doi/10.5555/1661445.1661605

[6] Chalupar, G., S. Peherstorfer, E. Poll and J. De Ruiter, *Automated reverse engineering using lego®*, in: *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*, 2014.
https://www.usenix.org/conference/woot14/workshop-program/presentation/chalupar

[7] Dahlqvist, F. and T. Schmid, *How to write a coequation ((co) algebraic pearls)*, in: *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
https://doi.org/10.4230/LIPIcs.CALCO.2021.13

[8] De Ruiter, J. and E. Poll, *Protocol state fuzzing of {TLS} implementations*, in: *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 193–206.
https://dl.acm.org/doi/10.5555/2831143.2831156

[9] Denis, F., A. Lemay and A. Terlutte, *Residual finite state automata*, in: *Annual Symposium on Theoretical Aspects of Computer Science*, Springer, 2001, pp. 144–157.
https://link.springer.com/chapter/10.1007/3-540-44693-1_13

[10] Feamster, N., J. Rexford and E. Zegura, *The road to sdn: an intellectual history of programmable networks*, ACM SIGCOMM Computer Communication Review **44** (2014), pp. 87–98.
https://doi.org/10.1145/2602204.2602219

[11] Foster, N., D. Kozen, K. Mamouras, M. Reitblatt and A. Silva, *Probabilistic netkat*, in: *European Symposium on Programming*, Springer, 2016, pp. 282–309.
https://link.springer.com/chapter/10.1007/978-3-662-49498-1_12

[12] Foster, N., D. Kozen, M. Milano, A. Silva and L. Thompson, *A coalgebraic decision procedure for netkat*, in: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 343–355. https://doi.org/10.1145/2676726.2677011

[13] Grathwohl, N. B. B., D. Kozen and K. Mamouras, *Kat+ b!*, in: *Proceedings of the joint meeting of the twenty-third EASCL annual conference on Computer Science Logic (CSL) and the twenty-ninth annual ACM/IEEE Symposium on Logic in Computer Science (LiCS)*, 2014, pp. 1–10.
https://doi.org/10.1145/2603088.2603095

[14] Hagerer, A., H. Hungar, O. Niese and B. Steffen, *Model generation by moderated regular extrapolation*, in: *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2002, pp. 80–95.
https://link.springer.com/chapter/10.1007/3-540-45923-5_6

[15] Howar, F., "Active learning of interface programs." Ph.D. thesis, Dortmund University of Technology (2012).
http://doi.org/10.17877/DE290R-4817

[16] Isberner, M., F. Howar and B. Steffen, *The TTT algorithm: a redundancy-free approach to active automata learning*, in: *International Conference on Runtime Verification*, Springer, 2014, pp. 307–322.
https://link.springer.com/chapter/10.1007/978-3-319-11164-3_26

[17] Kearns, M. J., U. V. Vazirani and U. Vazirani, "An introduction to computational learning theory," MIT press, 1994. ISBN 9780262111935.

[18] Kleene, S., *Representation of events in nerve nets and finite automata*, Automata studies **3** (1951), p. 41.
https://doi.org/10.1515/9781400882618-002

[19] Kozen, D., *A completeness theorem for Kleene algebras and the algebra of regular events*, Information and computation **110** (1994), pp. 366–390.
https://doi.org/10.1006/inco.1994.1037

[20] Kozen, D., *Kleene algebra with tests*, ACM Transactions on Programming Languages and Systems (TOPLAS) **19** (1997), pp. 427–443.
https://doi.org/10.1145/256167.256195

[21] Kozen, D., *Automata on guarded strings and applications*, Technical report, Cornell University (2001).
https://www.cs.cornell.edu/~kozen/Papers/ags.pdf

[22] Kozen, D., *On the coalgebraic theory of kleene algebra with tests*, in: *Rohit Parikh on Logic, Language and Society*, Springer, 2017 pp. 279–298.
https://doi.org/10.1007/978-3-319-47843-2_15

[23] Kozen, D. and F. Smith, *Kleene algebra with tests: Completeness and decidability*, in: *International Workshop on Computer Science Logic*, Springer, 1996, pp. 244–259.
https://link.springer.com/chapter/10.1007/3-540-63172-0_43

[24] Kozen, D. and W.-L. D. Tseng, *The Böhm–Jacopini theorem is false, propositionally*, in: *International Conference on Mathematics of Program Construction*, Springer, 2008, pp. 177–192.
https://doi.org/10.1007/978-3-540-70594-9_11

[25] Maler, O. and A. Pnueli, *On the learnability of infinitary regular sets*, Information and Computation **118** (1995), pp. 316–326.
https://doi.org/10.1006/inco.1995.1070

[26] Moerman, J., M. Sammartino, A. Silva, B. Klin and M. Szynwelski, *Learning nominal automata*, in: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 2017, pp. 613–625.
https://doi.org/10.1006/inco.1995.1070

[27] Moore, T., *Gedanken–experiments on Sequential Machines*, in: *Sequential Machines, Automata Studies, Annals of Mathematical Studies, no. 34*, Citeseer, 1956.
https://doi.org/10.1515/9781400882618-006

[28] Nerode, A., *Linear automaton transformations*, Proceedings of the American Mathematical Society **9** (1958), pp. 541–544.
https://doi.org/10.2307/2033204

[29] Pous, D., *Symbolic algorithms for language equivalence and kleene algebra with tests*, in: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 357–368.
https://doi.org/10.1145/2775051.2677007

[30] Rivest, R. L. and R. E. Schapire, *Inference of finite automata using homing sequences*, Information and Computation **103** (1993), pp. 299–347.
https://doi.org/10.1006/inco.1993.1021

[31] Schmid, T., T. Kappé, D. Kozen and A. Silva, *Guarded kleene algebra with tests: Coequations, coinduction, and completeness*, arXiv preprint (2021).
https://doi.org/10.48550/arXiv.2102.08286

[32] Smolka, S., N. Foster, J. Hsu, T. Kappé, D. Kozen and A. Silva, *Guarded kleene algebra with tests: verification of uninterpreted programs in nearly linear time*, Proceedings of the ACM on Programming Languages **4** (2019), pp. 1–28.
https://doi.org/10.1145/3371129

[33] Smolka, S., P. Kumar, D. M. Kahn, N. Foster, J. Hsu, D. Kozen and A. Silva, *Scalable verification of probabilistic networks*, in: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 190–203.
https://doi.org/10.1145/3314221.3314639

[34] Thompson, K., *Programming techniques: Regular expression search algorithm*, Communications of the ACM **11** (1968), pp. 419–422.
https://doi.org/10.1145/363347.363387

[35] Vaandrager, F., *Model learning*, Communications of the ACM **60** (2017), pp. 86–95.
https://doi.org/10.1145/2967606

[36] Vaandrager, F., B. Garhewal, J. Rot and T. Wißmann, *A new approach for active automata learning based on apartness*, arXiv preprint arXiv:2107.05419 (2021).
https://doi.org/10.48550/arXiv.2107.05419

[37] van Heerdt, G., M. Sammartino and A. Silva, *Calf: Categorical automata learning framework*, Computer Science Logic 2017 (2017).
https://drops.dagstuhl.de/opus/volltexte/2017/7695/pdf/LIPIcs-CSL-2017-29.pdf

[38] van Heerdt, G., M. Sammartino and A. Silva, *Learning automata with side-effects*, in: *International Workshop on Coalgebraic Methods in Computer Science*, Springer, 2020, pp. 68–89.
https://link.springer.com/chapter/10.1007/978-3-030-57201-3_5

[39] Zetzsche, S., G. van Heerdt, M. Sammartino and A. Silva, *Canonical automata via distributive law homomorphisms*, Electronic Proceedings in Theoretical Computer Science **351** (2021), p. 296–313.
http://doi.org/10.4204/EPTCS.351.18