CGE: A Sequential Learning Algorithm for Mealy Automata

Karl Meinke

School of Computer Science and Communication,
Royal Institute of Technology, 100-44 Stockholm, Sweden

Abstract. We introduce a new algorithm for sequential learning of Mealy automata by *congruence generator extension* (CGE). Our approach makes use of techniques from term rewriting theory and universal algebra for compactly representing and manipulating automata using finite congruence generator sets represented as *string rewriting systems* (SRS). We prove that the CGE algorithm correctly learns in the limit.

1 Introduction

Developments in software testing such as black-box checking [Peled et al. 1999], learning-based testing [Meinke 2004], adaptive model checking [Groce et al. 2006], and dynamic testing [Raffelt et al. 2008] motivate the need for new types of learning algorithms for automata. The common aim of these approaches to software testing is to model an unknown black-box system under test (SUT) as some form of automaton, and to dynamically learn this automaton and analyse its behavioural correctness using relevant test cases as membership queries and the SUT itself as the teacher.

A learning algorithm for automata is *sequential* if it can produce a sequence of hypothesis automata $\mathfrak{A}_0$, $\mathfrak{A}_1$, ... which are approximations to an unknown automata $\mathfrak{A}$, based on a sequence of information (queries and results) about $\mathfrak{A}$. This sequence should converge to a behaviourally equivalent automaton when given sufficient information about $\mathfrak{A}$. A sequential algorithm is *incremental* if computation of successive approximations can reuse previous results (e.g. equivalence relations on states). For further details see e.g. [Parekh and Honavar 2000]).

Our research [Meinke 2004] on learning-based testing has shown that sequential and incremental learning algorithms are crucial for the effectiveness of the above testing methods for four important reasons.

(1) Real software systems may be too big to be exactly or completely learned within a feasible timescale.

(2) Testing of specific system requirements or use cases typically does not require learning and analysis of the entire software system.

(3) The overhead of SUT execution to answer a membership query during learning may be non-neglegible compared with the execution time of the learning algorithm itself (see e.g. [Bohlin and Jonsson 2008]). Therefore membership queries should be seen as "expensive".

(4) Since membership queries are expensive, as many queries (i.e. test cases) as possible should be derived from the behavioural analysis of the hypothesis automaton, while as few queries as possible should be derived for reasons of internal book-keeping by the learning algorithm. Ideally, *every* membership query should represent a relevant and interesting test case.

In this paper we introduce a new sequential learning algorithm satisfying these four criteria. We prove that it correctly learns in the limit in the sense of [Gold 1967]. Since [Pao, Carr 1978] and [Dupont et al. 1994] it has been clear that automata learning can be viewed as a search problem in the lattice of automata solutions. We follow this philosophy, while changing it slightly to search among the lattice of all *consistent finitely generated congruences* on the prefix automaton. Hence, a significant contribution of our approach is to show how universal algebra (see e.g. [Meinke, Tucker 1993]) and term rewriting (see e.g. [Dershowitz, Jouannaud 1990]) can be applied to the learning problem. Using finitely generated congruences increases the efficiency of hypothesis automaton representation. Congruence construction is non-trivial in the case of Mealy automata since two interrelated congruences, on states and output values, must be learned. This makes the search space of solutions inherently more complex than the search space for learning a regular language acceptor. The CGE learning algorithm computes a pair of finite congruence generator sets, which is inherently more compact than an explicit state space partition for large automata. Furthermore, each hypothesis automaton can be simulated and statically analysed (for example by model checking) without ever explicitly constructing it as a quotient automaton. One can perform these operations using the generator sets alone.

In Section 1.1, we review some automata learning algorithms from the literature. In Section 2, we briefly review some elementary concepts from automata theory and universal algebra. In Section 3 we define and prove the correctness of a new *prefix completion algorithm* for SRS which is more efficient than the well known Knuth-Bendix completion algorithm [Knuth and Bendix 1970]. This algorithm efficiently constructs (a generator set for) the smallest state congruence containing a given set of generator pairs. In Section 4, we present the CGE learning algorithm and prove its correctness. In Section 5 we draw some conclusions, and in Appendix 1 we present a simple illustrative case study of CGE learning.

1.1 Sequential and Incremental Learning Algorithms for Automata

In [Dupont 1996], an incremental version RPNI2 of the RPNI learning algorithm of [Oncina and Garcia 1992] and [Lang 1992] is presented. The RPNI2 algorithm has some features in common with the CGE algorithm, but there are significant differences. Both RPNI2 and CGE perform a recursive depth first search of a lexicographically ordered state set with backtracking. However, RPNI2 is explicitly coded for Moore machines with binary (positive/negative) outputs (i.e. language recognition). Furthermore, RPNI2, like many other automata learning algorithms, represents and manipulates the hypothesis automaton state set by computing an equivalence relation on input strings. By contrast, CGE uses a purely symbolic approach based on finite congruence generator sets represented as string rewriting systems (SRS) that are used to compute normal forms of states. This latter representation is more compact because a congruence is an equivalence relation that is also closed under substitution. Several computational steps in RPNI2 such as quotient automaton derivation, and compatibility (consistency) checking by parsing, are unnecessary in the CGE algorithm, or are replaced by an efficient alternative based on string rewriting. Furthermore, RPNI2 computes a non-deterministic hypothesis automaton that is subsequently rendered deterministic, whereas CGE always maintains a deterministic hypothesis automaton by working directly with congruences. However, both RPNI2 and CGE learn in the limit. Neither algorithm needs to generate any new membership queries, other than those contained in the current query database, in order to produce a hypothesis automaton. Thus both RPNI2 and CGE fulfill criterion (4) of Section 1 above.

The IID algorithm of [Parekh et al. 1998] and the incremental learning algorithm introduced in [Porat, Feldman 1991] are also explicitly coded for positive/negative outcomes only (language recognition). Furthermore, because of internal book-keeping operations (see [Meinke and Sindhu 2010]) and lexicographic query ordering, these algorithms seem less efficient according to criterion (4).

2 Congruences, Generators and Quotient Automata

It is natural to model a Mealy automaton as a *many-sorted algebraic structure* (c.f. [Meinke, Tucker 1993]) by considering states and outputs as two separate types. In the sequel, $\Sigma = \{ \sigma_1, \ldots, \sigma_m \}$ is a fixed finite input alphabet, and $\Omega = \{ \omega_1, \ldots, \omega_n \}$ is a fixed finite output alphabet.

2.1. Definition. A *Mealy automaton* $\mathfrak{A}$ over Σ and Ω is a two-sorted algebra

$$\mathfrak{A} = \langle Q_{\mathfrak{A}},\ \Omega_{\mathfrak{A}},\ \delta_{\mathfrak{A}},\ \lambda_{\mathfrak{A}},\ q^0_{\mathfrak{A}},\ \omega_{1_A},\ \ldots,\ \omega_{n_A} \rangle$$

where the carrier set $Q_{\mathfrak{A}}$ is termed the *state set* of $\mathfrak{A}$ and the carrier set $\Omega_{\mathfrak{A}}$ is termed the *output set* of $\mathfrak{A}$. If the state set $Q_{\mathfrak{A}}$ is finite then $\mathfrak{A}$ is termed a *finite state Mealy automaton* otherwise $\mathfrak{A}$ is termed *infinite state.* Also

$$\delta_{\mathfrak{A}} : Q_{\mathfrak{A}} \times \Sigma \to Q_{\mathfrak{A}}, \qquad \lambda_{\mathfrak{A}} : Q_{\mathfrak{A}} \times \Sigma \to \Omega_{\mathfrak{A}}$$

are the *state transition function*, and *output function* respectively. Furthermore $q^0_{\mathfrak{A}} \in Q_{\mathfrak{A}}$ is the *initial state* and $\omega_{1_{\mathfrak{A}}},\ \ldots,\ \omega_{n_{\mathfrak{A}}} \in \Omega_{\mathfrak{A}}$ are *output constants* that interpret the output symbols $\omega_1, \ldots, \omega_n$ respectively. We let $MA(\Sigma,\ \Omega)$ denote the class of all Mealy automata over Σ and Ω.

2.2. Example. Define the *term*, *initial* or *absolutely free Mealy automaton* $T(\Sigma,\ \Omega) \in MA(\Sigma,\ \Omega)$ (which is infinite state) by:

$$Q_{T(\Sigma,\ \Omega)} = \Sigma^*, \qquad \Omega_{T(\Sigma,\ \Omega)} = \Sigma^+ \cup \Omega$$

then for any $\overline{\sigma} \in \Sigma^*$ and $\sigma \in \Sigma$, $\delta_{T(\Sigma,\ \Omega)}(\overline{\sigma},\ \sigma) = \lambda_{T(\Sigma,\ \Omega)}(\overline{\sigma},\ \sigma) = \overline{\sigma}\ .\ \sigma$. Also $q^0_{T(\Sigma,\ \Omega)} = \varepsilon$ and $\omega_{i_{T(\Sigma,\ \Omega)}} = \omega_i$ for $i = 1,\ \ldots,\ n$.

It is easily shown that $T(\Sigma,\ \Omega)$ is initial in the class $MA(\Sigma,\ \Omega)$, i.e. there exists a unique homomorphism $\phi : T(\Sigma,\ \Omega) \to \mathfrak{A}$ for each $\mathfrak{A} \in MA(\Sigma,\ \Omega)$. Hence every minimal or reachable Mealy automaton $\mathfrak{A} \in MA(\Sigma,\ \Omega)$ can be constructed as a quotient of $T(\Sigma,\ \Omega)$. This principle is applied in Section 4.

An *observation* for any Mealy automaton is defined to be a pair $(\overline{\sigma},\ \omega) \in \Sigma^+ \times \Omega$ consisting of an input sequence and the final output value. We define $lhs((\overline{\sigma},\ \omega)) = \overline{\sigma}$, and extend this operation to sets of obervations.

Recall the principles of quotient construction applied to Mealy automata.

2.3. Definition. Let $\mathfrak{A} \in MA(\Sigma,\ \Omega)$ be any Mealy automaton, and let $\equiv = \langle \equiv_Q,\ \ \equiv_\Omega \rangle$, where $\equiv_Q \subseteq Q_\mathfrak{A} \times Q_\mathfrak{A}$ is an equivalence relation on states and $\equiv_\Omega \subseteq \Omega_\mathfrak{A} \times \Omega_\mathfrak{A}$ is an equivalence relation on outputs.

We say that $\equiv$ is a *congruence* on $\mathfrak{A}$ if, and only if, $\equiv$ satisfies the following *substitutivity conditions.* For any $q,\ q' \in Q_\mathfrak{A}$ and $\sigma \in \Sigma$, if $q \equiv_Q q'$ then:

$$\delta_\mathfrak{A}(\ q,\ \sigma\ ) \equiv_Q \delta_\mathfrak{A}(\ q',\ \sigma\ ),\ \ \textit{and}\ \ \lambda_\mathfrak{A}(\ q,\ \sigma\ ) \equiv_\Omega \lambda_\mathfrak{A}(\ q',\ \sigma\ ).$$

In this case, $\equiv_Q$ is termed a *state congruence* and $\equiv_\Omega$ is termed an *output congruence.*

Let $X\ = \langle\ X_Q,\ X_\Omega \rangle$ be a pair of binary relations $X_Q \subseteq Q_\mathfrak{A} \times Q_\mathfrak{A}$ and $X_\Omega \subseteq \Omega_\mathfrak{A} \times \Omega_\mathfrak{A}$ then X *generates* $\equiv$ if and only if, $\equiv$ is the smallest congruence on $\mathfrak{A}$ containing X, and $\equiv$ is *finitely generated* if X_Q and X_Ω are both finite.

Define the *quotient automaton* $\mathfrak{A}/\equiv\ \in MA(\Sigma,\ \Omega)$ by

$$Q_{\mathfrak{A}/\equiv} = Q_\mathfrak{A}/{\equiv_Q}, \qquad \Omega_{\mathfrak{A}/\equiv} = \Omega_\mathfrak{A}/{\equiv_\Omega}$$

and for any $q \in Q_\mathfrak{A}$ and $\sigma \in \Sigma$,

$$\delta_{\mathfrak{A}/\equiv}(\ q/{\equiv_Q},\ \sigma\ ) = \delta_\mathfrak{A}(\ q,\ \sigma\ )/{\equiv_Q}, \qquad \lambda_{\mathfrak{A}/\equiv}(\ q/{\equiv_Q},\ \sigma\ ) = \lambda_\mathfrak{A}(\ q,\ \sigma\ )/{\equiv_\Omega}.$$

Also $q^0_{\mathfrak{A}/\equiv} = q^0_\mathfrak{A}/{\equiv_Q}$ and $\omega_{i_{\mathfrak{A}/\equiv}} = \omega_{i_\mathfrak{A}}/{\equiv_\Omega}$ for $1 \le i \le n$.

For black-box testing and many other applications, it suffices to learn an unknown automaton up to *behavioural equivalence.*

2.4. Definition. Let $\mathfrak{A},\ \mathfrak{B} \in MA(\Sigma,\ \Omega)$ be any Mealy automata, and let $\equiv^\phi$ and $\equiv^\psi$ be the kernels of the unique homomorphisms $\phi : T(\Sigma,\ \Omega) \to \mathfrak{A}$ and $\psi : T(\Sigma,\ \Omega) \to \mathfrak{B}$ respectively. We say that $\mathfrak{A}$ and $\mathfrak{B}$ are *behaviourally equivalent* if, and only if, $\equiv^\phi_\Omega = \equiv^\psi_\Omega$.

Intuitively, two Mealy automata $\mathfrak{A}$ and $\mathfrak{B}$ are behaviourally equivalent if $\mathfrak{A}$ and $\mathfrak{B}$ always produce the same output sequences given the same input sequence.

3 String Rewriting Systems and Rule Completion

In this section we consider the concept of a *confluent terminating string rewriting system* (SRS) which not only provides a compact representation of a congruence, but also comes with a natural model of computation known as *string rewriting.* This allows us to directly simulate computations performed by quotient automata while avoiding their explicit construction. String rewriting is a special case of the more general theory of term rewriting ([Dershowitz, Jouannaud 1990]). We borrow important concepts from this theory such as the *completion* of a set of rewrite rules. Our main result in this section is to define and prove the correctness of the *prefix completion algorithm.* This auxiliary algorithm plays an important role for efficient congruence construction and consistency checking in the CGE algorithm of Section 4.

3.1. Definition

(i) A *string rewriting rule* over Σ is a pair $(l,\ r) \in \Sigma^* \times \Sigma^*$. Often we use the more intuitive notation $l \rightarrow r$ to denote the rule $(l,\ r)$. By a *string rewriting system* (SRS) over Σ we mean a set $R \subseteq \Sigma^* \times \Sigma^*$ of string rewriting rules.

(ii) Let $\rho = (l,\ r)$ be a string rewriting rule and let $\overline{\sigma},\ \overline{\sigma'} \in \Sigma^*$ be any strings. We say that $\overline{\sigma}$ *rewrites to* $\overline{\sigma'}$ using ρ and write $\overline{\sigma} \xrightarrow{\rho} \overline{\sigma'}$ if, and only if for some $\overline{\sigma_0} \in \Sigma^*$ and $\overline{\sigma} = l\ .\ \overline{\sigma_0}$, i.e. l is a prefix of $\overline{\sigma}$ and, $\overline{\sigma'} = r\ .\ \overline{\sigma_0}$.

(iii) If $R \subseteq \Sigma^* \times \Sigma^*$ is an SRS then we say that $\overline{\sigma}$ *rewrites to* $\overline{\sigma'}$ using R *in one step* and write $\overline{\sigma} \xrightarrow{R} \overline{\sigma'}$ if, and only if, for some $\rho \in R$, $\overline{\sigma} \xrightarrow{\rho} \overline{\sigma'}$.

(iv) We let $\xrightarrow{R^*} \subseteq \Sigma^* \times \Sigma^*$ denote the reflexive transitive closure of $\xrightarrow{R}$. We define the *bi-rewriting relation* $\xleftrightarrow{R^*} \subseteq \Sigma^* \times \Sigma^*$ for any $\overline{\sigma},\ \overline{\sigma'} \in \Sigma^*$, by:

$$\overline{\sigma} \xleftrightarrow{R^*} \overline{\sigma'} \ \Leftrightarrow\ \exists \overline{\sigma_0} \in \Sigma^* \ \textit{such that}\ \overline{\sigma} \xrightarrow{R^*} \overline{\sigma_0}\ \textit{and}\ \overline{\sigma'} \xrightarrow{R^*} \overline{\sigma_0}.$$

Any rewrite sequence can be shown to terminate if it consists of a sequence of strings that can be shown to be strictly decreasing according to some well-ordering, e.g. the short-lex ordering on strings.

3.2. Definition
(i) Let $D \subseteq \Sigma \times \Sigma$ be a linear ordering on Σ, and let $\leq^D \subseteq \Sigma^n \times \Sigma^n$ for $n \geq 1$ be the induced *lexicographic ordering.* Define the *short-lex ordering* $\leq^D \subseteq \Sigma^* \times \Sigma^*$ by $\sigma_1,\ \ldots,\ \sigma_m \leq^D \sigma'_1,\ \ldots,\ \sigma'_n \Leftrightarrow m < n$ or $m = n$ and $\sigma_1,\ \ldots,\ \sigma_m \leq^D \sigma'_1,\ \ldots,\ \sigma'_n$. Then $\leq^D$ is a well-ordering on finite strings. Our main purpose for introducing this ordering is to prove termination of rewrite sequences by showing that the individual strings in a sequence are well-ordered.

3.3. Definition. Let $l \rightarrow r \in \Sigma^* \times \Sigma^*$ be a rewrite rule. We say that $l \rightarrow r$ is *reducing* if, and only if, $r <^D l$. If $R \subseteq \Sigma^* \times \Sigma^*$ is an SRS then R is *reducing* if, and only if, every rule $\rho \in R$ is reducing.

3.4. Proposition. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a reducing SRS. Then R is terminating, i.e. there does not exist an infinite sequence $\overline{\sigma_0},\ \overline{\sigma_1},\ \ldots \in \Sigma^*$ of strings such that for each $i \geq 0$, there exists $\rho_i \in R$ for which $\overline{\sigma_i} \xrightarrow{\rho_i} \overline{\sigma_{i+1}}$.*

Proof. Immediate since $\leq^D$ is a well ordering on Σ^*.

Algorithm 1. Prefix Completion Algorithm

```
Function C(R)
    $unfinished \leftarrow true$
    while $unfinished$ do
        Compute the set $Max_{\preceq}(R)$ of all maximal lhs terms
        foreach $l \in Max_{\preceq}(R)$ do
            construct the tower $T(l)$ under $l$
        $unfinished \leftarrow false$
        foreach $l \in Max_{\preceq}(R)$ do
            // Process the first block in $T(l)$
            // Collect all rhs terms from rules in block $T(l)(1)$
            for $j \leftarrow 2$ to $|T(l)(1)|$ do
            $Rhs\_Terms \leftarrow Rhs\_Terms \cup \{ T(l)(i)(j)_2 \}$
            // Get the minimum rhs term in block $T(l)(1)$
            $rhs\_min \leftarrow T(l)(1)(1)_2$
            // Orient all critical pairs as new rules.
            forall $t \in Rhs\_Terms$ do
                if $t \rightarrow rhs\_min \notin R$ then  $R \leftarrow R \cup \{ t \rightarrow rhs\_min \}$
                $unfinished \leftarrow true$
            // Process all remaining blocks in $T(l)$
            for $i \leftarrow 2$ to $|T(l)|$ do
                // Collect all rhs terms from rules in block $T(l)(i)$
                for $j \leftarrow 2$ to $|T(l)(i)|$ do
                    $Rhs\_Terms \leftarrow Rhs\_Terms \cup \{ T(l)(i)(j)_2 \}$
                // Complete the previous minimum rhs term
                $rhs\_min\_completed \leftarrow$
                    $concat(rhs\_min, (T(l)(i)(1)_1 - T(l)(i-1)(1)_1))$
                // Compute new value of rhs_min and update Rhs_Terms
                if $T(l)(i)(1)_2 \leq^D rhs\_min\_completed$ then
                    $rhs\_min \leftarrow T(l)(i)(1)_2$
                    if $T(l)(i)(1)_2 \neq rhs\_min\_completed$ then
                        $Rhs\_Terms \leftarrow Rhs\_Terms \cup \{ rhs\_min\_completed \}$
                else
                    $rhs\_min \leftarrow rhs\_min\_completed$
                    $Rhs\_Terms \leftarrow Rhs\_Terms \cup \{ T(l)(i)(1)_2 \}$
                // Orient all critical pairs as new rules
                forall $t \in Rhs\_Terms$ do
                    if $t \rightarrow rhs\_min \notin R$ then  $R \leftarrow R \cup \{ t \rightarrow rhs\_min \}$
                    $unfinished \leftarrow true$
    return $R$
end
```

In general, the order in which rewrite rules are applied can influence the outcome. An SRS which does not depend on the order of application of rules is termed a *confluent* SRS.

3.5. Definition. Let $R \subseteq \Sigma^* \times \Sigma^*$ be an SRS.

(i) We say that R is *confluent* if, and only if, for any $\overline{\sigma_0},\ \overline{\sigma_1},\ \overline{\sigma_2} \in \Sigma^*$, if $\overline{\sigma_0} \xrightarrow{R^*} \overline{\sigma_1}$ and $\overline{\sigma_0} \xrightarrow{R^*} \overline{\sigma_2}$ then there exists $\overline{\sigma} \in \Sigma^*$ such that $\overline{\sigma_1} \xrightarrow{R^*} \overline{\sigma}$ and $\overline{\sigma_2} \xrightarrow{R^*} \overline{\sigma}$.

(ii) We say that R is *locally confluent* if, and only if, for any $\overline{\sigma_0},\ \overline{\sigma_1},\ \overline{\sigma_2} \in \Sigma^*$, if $\overline{\sigma_0} \xrightarrow{R} \overline{\sigma_1}$ and $\overline{\sigma_0} \xrightarrow{R} \overline{\sigma_2}$ then there exists $\overline{\sigma} \in \Sigma^*$ such that $\overline{\sigma_1} \xrightarrow{R^*} \overline{\sigma}$ and $\overline{\sigma_2} \xrightarrow{R^*} \overline{\sigma}$.

Clearly confluence implies local confluence. The converse comes from a celebrated general result in term rewriting theory.

3.6. Newman's Lemma. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a reducing SRS. Then R is locally confluent if, and only if, R is confluent.*

Non-confluent SRS are undesirable since the resulting equivalence classes lack unique normal forms. Fortunately, they can sometimes be converted to an equivalent confluent SRS by *completion*, i.e. conservatively adding extra rewrite rules that rectify divergent critical pairs of rules. We introduce an efficient algorithm that completes R by adding an additional set of rules that grows only linearly in the size of the problem. For this purpose we use a specific well-ordering on rewrite rules. Let $\preceq\ \subseteq \Sigma^* \times \Sigma^*$ be the prefix ordering on strings.

3.7. Definition. Define the *tower ordering* $\leq^{T(D)}\ \subseteq (\Sigma^* \times \Sigma^*)^2$ for any $\rho_1 = l_1 \rightarrow r_1$ and $\rho_2 = l_2 \rightarrow r_2$ by $\rho_1 \leq^{T(D)} \rho_2 \ \Leftrightarrow\ l_1 \prec l_2\ \ or\ \ l_1 = l_2\ \ and\ \ r_1 \leq^D r_2$.

Next we introduce an abstract data structure for storing rewrite rules in such a way that we can conveniently access exactly those pairs of rules that yield critical pairs.

3.8. Definition

(i) Let $R \subseteq \Sigma^* \times \Sigma^*$ be any finite SRS. Define $Max_{\preceq}(R)$ to be the set of all *maximal left hand sides of rules* in R under the prefix ordering, i.e.

$$Max_{\preceq}(R) =$$

$$\{\ l \in \Sigma^* \ \mid\ l \rightarrow r \in R \ \ and\ \ there\ \ is\ \ no\ \ l' \rightarrow r' \in R\ \ such\ \ that\ \ l \prec l'\ \}.$$

(ii) For each maximal lhs $l \in Max_{\preceq}(R)$ define the *tower of rules* $T(l) \in ((\Sigma^* \times \Sigma^*)^+)^+$ *under* l to be the finite sequence of finite sequences of all rewrite rules $l' \rightarrow r' \in R$ such that $l' \preceq l$ ordered according to the following conditions.

Let us term the ith finite sequence $T(l)(i)$ the *ith block* in $T(l)$. Then each block $T(l)(i)$ must satisfy the following properties:

(ii.a) All rules in the same block $T(l)(i)$ have the same lhs, i.e. for all $1 \leq i \leq |T(l)|$, and all $1 \leq j,\ k \leq |T(l)(i)|$, $T(l)(i)(j)_1 = T(l)(i)(k)_1$.

(ii.b) All blocks are strictly linearly ordered by the prefix ordering on their unique lhs, i.e. for all $1 \leq i,\ j \leq |T(l)|$, $i < j$ implies $T(l)(i)(1)_1 \prec T(l)(j)(1)_1$.

(ii.c) All rewrite rules in the same block $T(l)(i)$ are strictly linearly ordered by the short-lex ordering $\leq^D$ on their rhs, i.e. for all $1 \leq i \leq |T(l)|$ and for all $1 \leq j,\ k \leq |T(l)(i)|$, $j < k$ implies $T(l)(i)(j)_2 <^D T(l)(i)(k)_2$.

3.9. Prefix Completion Algorithm
Define the *prefix completion function*

$$C : \wp^{fin}(\Sigma^* \times \Sigma^*) \rightarrow \wp^{fin}(\Sigma^* \times \Sigma^*)$$

constructively by Algorithm 1.

3.10. Correctness Theorem. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be any finite reducing SRS. The prefix completion algorithm terminates given input R and $C(R)$ is a finite confluent reducing SRS. Also the bi-rewriting relation $\overset{C(R)^*}{\longleftrightarrow} \subseteq \Sigma^* \times \Sigma^*$ is the smallest state congruence on the term automaton $T(\Sigma, \ \Omega)$ that contains R.*

Proof. Exercise.

The most important property of a confluent reducing SRS (used in Consistency Algorithm 4.2) is that it yields a unique normal form for every string.

3.11. Definition. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a confluent reducing SRS. For any $\overline{\sigma} \in \Sigma^*$ we define the *normal form* $Norm_R(\overline{\sigma}) \in \Sigma^*$ of $\overline{\sigma}$ (modulo R) to be the unique irreducible string obtained by rewriting $\overline{\sigma}$ using R.

To construct a quotient automaton it is not enough to have a state congruence, we also need an output congruence.

3.12. Definition. Let $\equiv \subseteq \Sigma^* \times \Sigma^*$ be any binary relation on Σ^*. Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be a set of observations. Define the binary relation $\equiv^\Lambda \subseteq (\Sigma^+ \cup \Omega)^2$ by

$$\equiv^\Lambda = RST(\ \Lambda \cup \{\ (\overline{\sigma}\ .\ \sigma,\ \overline{\sigma'}\ .\ \sigma)\ \mid\ (\overline{\sigma},\ \overline{\sigma'}) \in \equiv \ \ and\ \ \sigma \in \Sigma\ \}\ ),$$

where $RST(X)$ is the reflexive symmetric transitive closure of X.

3.13. Proposition. *Let $\equiv \subseteq \Sigma^* \times \Sigma^*$ be any state congruence on $T(\Sigma, \ \Omega)$. Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be a set of observations. Then $\equiv^\Lambda$ is the smallest output congruence on the term automaton $T(\Sigma, \ \Omega)$ that contains Λ.*

Proof. Exercise.

4 Learning by Congruence Generator Extension (CGE)

In this section we introduce the CGE learning algorithm for Mealy automata and prove its correctness. The basic idea of our method is to sequentially construct a sequence of hypothesis automata $H_1, \ H_2, \ \ldots$ based on the results of a series of observations $o_1, \ o_2, \ \ldots$ about an unknown Mealy automaton $\mathfrak{A}$. We represent each hypothesis automaton H_i as a quotient term automaton:

$$H_i = T(\Sigma, \ \Omega)/ \equiv^i, \quad i = 1, \ 2, \ \ldots,$$

and the sequence will eventually converge to a quotient automaton H_n that is behaviorally equivalent to $\mathfrak{A}$ (Theorem 4.7), and has a minimised state space. Each congruence $\equiv^i$, for $i = 1, \ 2, \ \ldots$, is constructed from the current finite observation set $\Lambda_i = \{ o_1, o_2, \ldots, o_i \}$. We identify bounds on n and Λ_n which guarantee convergence, in terms of the size and structure of $\mathfrak{A}$.

Key features of the CGE learning algorithm are the following:

(i) Learning in the limit is achieved in the sense of [Gold 1967].

(ii) The number of observations between successive hypothesis automata constructions H_i and H_{i+1} can be as small as one observation o_{i+1}. (Recall Section 1 criterion (4).)

(iii) There is always complete freedom to choose a new observation o_{i+1}. For example, new observations may be made randomly, or from static analysis of the current hypothesis H_i against a requirement specification.

(iv) The hypothesis automaton H_i is unambiguously represented by finite congruence generator sets and never explicitly constructed.

Let us begin by formalising some notions of consistency.

4.1. Definition

(i) Let $\equiv \subseteq (\Sigma^+ \cup \Omega)^2$ be an equivalence relation on outputs. We say that $\equiv$ is *consistent* if and only if, for any output symbols $\omega, \omega' \in \Omega$ $\omega \neq \omega' \Rightarrow \omega \not\equiv \omega'$.

(ii) Let $\equiv \subseteq \Sigma^* \times \Sigma^*$ be any relation and $\Lambda \subseteq \Sigma^+ \times \Omega$ be any set of observations. Then $\equiv$ is said to be *Λ-consistent* if, and only if $\equiv^{\Lambda}$ is consistent

(iii) Let $R \subseteq \Sigma^* \times \Sigma^*$ be a confluent reducing SRS then R is said to be *Λ-consistent* if and only if, the bi-rewriting relation $\xleftrightarrow{R^*}$ is Λ-consistent.

(iv) Let $\mathfrak{A} \in MA(\Sigma, \Omega)$ be any automaton. Then $\mathfrak{A}$ is said to be *consistent* if, and only if, for any output symbols $\omega, \omega' \in \Omega$ $\omega \neq \omega' \Rightarrow \omega_{\mathfrak{A}} \neq \omega'_{\mathfrak{A}}$.

During CGE we build up each SRS R^i (which represents $\equiv^i$) sequentially from the empty set by checking new rules for consistency.

4.2. Consistency Algorithm. Define the *consistency function*

$$Cons \ : \wp^{fin}(\Sigma^* \times \Sigma^*) \times \wp^{fin}(\Sigma^+ \times \Omega) \times \Sigma^* \times \Sigma^* \to \{ \mathit{true}, \mathit{false} \}$$

for any SRS $R \subseteq \Sigma^* \times \Sigma^*$, observation set $\Lambda \subseteq \Sigma^+ \times \Omega$ and strings $\overline{\sigma}, \overline{\sigma'} \in \Sigma^*$ by:

$Cons\ (R, \Lambda, , \overline{\sigma}, \overline{\sigma'}) =$ let $S = C(R \cup \{ \overline{\sigma} \to \overline{\sigma'} \})$ in

$$\begin{cases} \mathit{false} & \text{if } Norm_S(\sigma_1, \ldots, \sigma_m) = Norm_S(\sigma'_1, \ldots, \sigma'_n), \\ & \text{for some } (\sigma_1, \ldots, \sigma_m, \sigma, \omega) \in \Lambda \\ & \text{and } (\sigma'_1, \ldots, \sigma'_n, \sigma, \omega') \in \Lambda \text{ with } \omega \neq \omega'. \\ \mathit{true} & \text{otherwise.} \end{cases}$$

4.3. Correctness Theorem. *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a finite confluent reducing SRS and let $\Lambda \subseteq \Sigma^+ \times \Omega$ be any set of observations. For any $\overline{\sigma}, \overline{\sigma'} \in \Sigma^*$ such that $\overline{\sigma'} <^D \overline{\sigma}$, $Cons\ (R, \Lambda, , \overline{\sigma}, \overline{\sigma'}) = true$ if, and only if $S = C(R \cup \{ \overline{\sigma} \to \overline{\sigma'} \})$ is a Λ-consistent SRS.*

Proof. Exercise, using Definition 3.13.

The CGE algorithm takes as input a set Λ of observations, an SRS R which is under construction, a finite sequence $A = A_1, \ldots, A_l$ of strings $A_i \in \Sigma^*$ (state representations) which is strictly linearly ordered by $<^D$, and two indices m, n within $1, \ldots, l$ for the lhs and rhs of a putative reducing rewrite rule $A_m \to A_n$. The indices m and n are recursively incremented to traverse A in a specific order, starting from their initial values. The basic idea is first to check that the putative rule $A_m \to A_n$ is not already subsumed by other rules in R (i.e. redundant). Assuming that it is not redundant, then we add the putative rule $A_m \to A_n$ to R and complete. If the resulting confluent reducing SRS $C(\ R \cup \{\ A_m \to A_n\ \}\ )$ is Λ-consistent then the rule is retained, otherwise it is discarded. In both cases, we recursively proceed to the next rule in A until we have traversed A. The strict linear ordering of A by $<^D$ ensures that $A_m \to A_n$ is always a reducing rule whenever $m > n$.

4.4. CGE Learning Algorithm.
Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be a given finite set of observations. Define the *Congruence Generator Extension function*

$$CGE_\Lambda : \wp^{fin}(\Sigma^* \times \Sigma^*) \times (\Sigma^*)^+ \times \mathbb{N} \times \mathbb{N} \to \wp^{fin}(\Sigma^* \times \Sigma^*)$$

constructively for any SRS $R \subseteq \Sigma^* \times \Sigma^*$, any finite sequence $A = A_1, \ldots, A_l \in (\Sigma^*)^+$ of strings, and any indices $m,\ n \in \mathbb{N}$, by Algorithm 2. Define the *congruence generator extension function* $CGE_\Lambda(A) = CGE_\Lambda(\ \emptyset,\ A,\ 2,\ 1\ )$.

4.5. Proposition. *Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be any finite set of observations on a consistent automaton $\mathfrak{A} \in MA(\Sigma,\ \Omega)$. Let $A \in (\Sigma^*)^+$ be a sequence of length 2 or more ($|A| \geq 2$) of strings , in strictly ascending $<^D$ order. Then $R = CGE_\Lambda(A)$ is Λ-consistent confluent and reducing SRS, and hence $\overset{R^*}{\longleftrightarrow}$ is a state congruence on $T(\Sigma,\ \Omega)$.*

Proof. By induction.

The key insight for proving the correctness of CGE learning is to establish that all loops in the path structure of an unknown automaton $\mathfrak{A}$ will be correctly learned by executing a minimum number of observations on $\mathfrak{A}$. This number is bounded by the maximum loop size of $\mathfrak{A}$. So under an appropriate observation strategy the sequence of hypothesis automata generated by CGE learning eventually converges in terms of overall state space size.

4.6. Convergence Theorem. *Let $\mathfrak{A} \in MA(\Sigma,\ \Omega)$ be a consistent automaton. Let n be the length of the longest acyclic path in $\mathfrak{A}$. Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be any set of observations on $\mathfrak{A}$ that contains all observations of length $2n+1$ or less. Let Λ^+ be the prefix closure of all inputs in Λ, and let $\Lambda^+_{<^D}$ be the sequence of all members of Λ^+ in strictly ascending $<^D$ order. Let $R = CGE_\Lambda(\Lambda^+_{<^D}\ )$. For every $\overline{\sigma} \in \Sigma^*$ there exists $\overline{\sigma'} \in \Sigma^*$ such that $\overline{\sigma} \xrightarrow{R^*} \overline{\sigma'}$ and $|\overline{\sigma'}| \leq n$.*

Proof. By induction on the well ordering $\leq^D$.

Algorithm 2. CGE Algorithm

```
Function $CGE_\Lambda$( $R$, $A$, $m$, $n$ )
    // Check subsumption and consistency of rule $A_m \rightarrow A_n$
    if $A_m \overset{R^*}{\not\longrightarrow} A_n$ and Cons ( $R$, $\Lambda$, $A_m$, $A_n$ ) then
        // Add rule $A_m \rightarrow A_n$ to $R$
        if $n = m - 1$ and $m < |A|$ then
            return $CGE_\Lambda$( $C$( $R \cup \{ A_m \rightarrow A_n \}$ ), $A$, $m+1$, $1$ )
        else
            if $n < m - 1$ then
                return $CGE_\Lambda$( $C$( $R \cup \{ A_m \rightarrow A_n \}$ ), $A$, $m$, $n+1$ )
            else
                // Finished traversal of $A$
                return $C$( $R \cup \{ A_m \rightarrow A_n \}$ )
    else
        // Don't add rule $A_m \rightarrow A_n$ to $R$
        if $n = m - 1$ and $m < |A|$ then
            return $CGE_\Lambda$( $R$, $A$, $m+1$, $1$ )
        else
            if $n < m - 1$ then
                return $CGE_\Lambda$( $R$, $A$, $m$, $n+1$ )
            else
                // Finished traversal of $A$
                return R
end
```

From Convergence Theorem 4.6 we can conclude that the state space size of each generated hypothesis automaton is bounded from above by $|\Sigma|^n$, where n is the length of longest acyclic path in $\mathfrak{A}$. Our main result is the following theorem.

4.7. Correctness Theorem. *Let $\mathfrak{A} \in MA(\Sigma,\ \Omega)$ be a consistent automaton. Let n be the length of the longest acyclic path in $\mathfrak{A}$. Let $\Lambda \subseteq \Sigma^+ \times \Omega$ be any set of observations on $\mathfrak{A}$ that contains all observations of length $2n+1$ or less. Let $R = CGE_\Lambda(\Lambda^+_{<_D}\ )$. Then $\mathfrak{A}$ and the quotient term automaton $T(\Sigma,\ \Omega)/(\ \overset{R^*}{\longleftrightarrow}\ ,\ \overset{R^*\Lambda}{\longleftrightarrow}\ )$ are behaviourally equivalent.*

Proof. Induction using Theorem 4.6, Definitions 2.4 and 3.12.

As we indicated at the start of this section, the CGE function presented in Algorithm 2 is applied iteratively to a sequence of input/output observations on an unknown automaton $\mathfrak{A}$. Then Correctness Theorem 4.7 establishes that when this sequence of observations is sufficiently large then the sequence of hypothesis automaton indeed converges to $\mathfrak{A}$ (up to behavioural equivalence). This iterative method is defined more precisely in Algorithm 3. In line 6 we compute the R normal form of the input string of each observation in the prefix closure $PrefObs((\overline{\sigma}_i, \omega_i))$ of observation $(\overline{\sigma}_i, \omega_i)$, which consists of all prefixes of $\overline{\sigma}_i$ and their respective outputs. Taking the prefix closure speeds up average convergence.

Algorithm 3. CGE Iteration Algorithm

```
1 Input: A sequence $S = (\overline{\sigma}_1, \omega_1),\ \ldots,\ (\overline{\sigma}_n, \omega_n)$ of $n$ observations of the I/O
  behavior of $\mathfrak{A}$, where $(\overline{\sigma}_i, \omega_i) = (\sigma_i^1,\ \ldots,\ \sigma_i^{k(i)},\ \omega_i) \in \Sigma^+ \times \Omega$.
2 Output: A sequence $(R_i^{state}, R_i^{output})$ for $i = 1,\ \ldots,\ n$ of congruence generator
  sets for quotient Mealy machines $M_i$ represented as SRS.
   1. begin
   2. //Perform Initialization
   3. $\Lambda = \emptyset$, $\Lambda^R = \emptyset$, $R = \emptyset$, $A = \emptyset$, $i = 1$,
   4. while $i \leq n$ do
   5.     //Normalise the $i$-th observation and all its prefix observations with $R$
   6.     $norm = Norm_R(\ PrefObs((\overline{\sigma}_i, \omega_i))\ )$
   7.
   8.     if $norm \not\subseteq \Lambda^R$ then
   9.         //At least one prefix observation of $i$-th observation $(\overline{\sigma}_i, \omega_i)$
  10.         //has no equivalent in $\Lambda^R$.
  11.         //So update $\Lambda$, $R$ and $\Lambda^R$. This will also resolve inconsistency
  12.         //if some prefix of $(\overline{\sigma}_i, \omega_i)$ is inconsistent with $R$.
  13.         $\Lambda = \Lambda \cup \{\ PrefObs((\overline{\sigma}_i, \omega_i))\ \}$
  14.         $A = lhs(\Lambda) \cup \{\ \overline{\tau}\sigma_0 \mid \exists\ \overline{\sigma} \in lhs(\Lambda),\ \overline{\tau} \prec \overline{\sigma},\ \sigma_0 \in \Sigma\ \}$
  15.         $R = CGE_\Lambda(\ A_{<^D}\ )$
  16.         $\Lambda^R = Norm_R(\Lambda)$
  17.
  18.     Print( $i$; $(\overline{\sigma}_i, \omega_i)$; $(R,\ \{\ l\sigma \to r\sigma \mid l \to r \in R,\ \sigma \in \Sigma\ \} \cup \Lambda^R)$ )
  19.     $i = i + 1$
```

Notice also that the input sequence is pruned (line 8), so that only observations that give rise to a new hypothesis automaton are accepted and integrated. On line 14 we extend the set Λ of input strings with all one element extensions of proper prefixes. This step ensures that the quotient automaton will be finite (c.f. the kernel construction in [Dupont 1996]). For convenience, the sequence of constructed hypothesis automata, each of which is represented as a pair of SRS, is buffered in a print statement (line 18). To illustrate the principles of our approach, in Appendix 1 we give a small case study of the sequence of SRS produced when learning a simple Mealy machine using CGE.

5 Conclusions

We have introduced the CGE algorithm for sequential learning of Mealy automata. This algorithm efficiently represents and manipulates learned hypothesis automata using pairs of finite generator sets for state and output congruences represented as string rewriting systems (SRS). We have developed the prefix completion algorithm for efficient SRS representation. We have shown that CGE correctly learns in the limit. Elsewhere we will present detailed proofs and show that it computes the minimimum state space.

The simple CGE algorithm presented here is rich in optimisations that can be applied, though the proof of correctness becomes progressively more complex. We will explore such optimisations in future research. For example, this algorithm is not yet incremental as one would like, since the SRS set R is fully recomputed each time a new observation is read, even if it is consistent with the current set R. It seems possible to reduce this computation and avoid recomputing the whole of R. Nevertheless, computing a monotone increasing sequence of SRS is fundamentally impossible, since each SRS rule is a current hypothesis about $\mathfrak{A}$ that might possibly be rejected later. So purely incremental learning is generally not possible.

We gratefully acknowledge assistance from Mr Niu Fei in the construction of a Python implementation of CGE, which was used for initial benchmarking studies. Financial support for this research came from the Swedish Research Council (VR) and the European Union under project HATS FP7-231620.

References

[Balcazar et al. 1997] Balcazar, J.L., Diaz, J., Gavalda, R.: Algorithms for learning finite automata from queries: a unified view. In: Advances in Algorithms, Languages and Complexity, pp. 53–72. Kluwer, Dordrecht (1997)

[Bohlin and Jonsson 2008] Bohlin, T., Jonsson, B.: Regular Inference for Communication Protocol Entities, Tech. Report 2008-024, Dept. of Information Technology, Uppsala University (2008)

[Dershowitz, Jouannaud 1990] Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. North Holland, Amsterdam (1990)

[Dupont 1996] Incremental regular inference. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS(LNAI), vol. 1147, pp. 222–237. Springer, Heidelberg (1996)

[Dupont et al. 1994] What is the search space of the regular inference? In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 25–37. Springer, Heidelberg (1994)

[Gold 1967] Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)

[Groce et al. 2006] Groce, A., Peled, D., Yannakakis, M.: Adaptive Model Checking. Logic Journal of the IGPL 14(5), 729–744 (2006)

[Knuth and Bendix 1970] Knuth, D.E., Bendix, P.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–269. Pergamon Press, Oxford (1970)

[Lang 1992] Lang, K.J.: Random DFA's can be approximately learned from sparse uniform examples. In: Proc. of 5th ACM workshop on Computational Learning Theory, pp. 45–52 (1992)

[Meinke 2004] Meinke, K.: Automated Black-Box Testing of Functional Correctness using Function Approximation. In: Rothermel, G. (ed.) Proc. ACM SIGSOFT Int. Symp. on Software Testing and Analysis, ISSTA 2004. Software Engineering Notes, vol. 29 (4), pp. 143–153. ACM Press, New York (2004)

[Meinke and Sindhu 2010] Meinke, K., Sindhu, M.: On the Correctness and Performance of the IID Incremental Learning Algorithm for DFA, technical report, School of Computer Science and Communication, Royal Institute of Technology, Stockholm (2010)

[Meinke, Tucker 1993] Meinke, K., Tucker, J.V.: Universal Algebra. In: Abramsky, S., Gabbay, D., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 1, pp. 189–411. Oxford University Press, Oxford (1993)

[Oncina and Garcia 1992] Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Perez de la Blanca, N., Sanfeliu, A., Vidal, E. (eds.) Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)

[Parekh and Honavar 2000] Parekh, R., Honavar, V.: Grammar inference, automata induction and language acquisition. In: Dale, Moisl, Somers (eds.) Handbook of Natural Language Processing. Marcel Dekker, New York

[Pao, Carr 1978] A solution of the syntactic induction-inference problem for regular languages. Computer languages 3, 53–64 (1978)

[Parekh et al. 1998] Parkeh, R.G., Nichitiu, C., Honavar, V.G.: A polynomial time incremental algorithm for regular grammar inference. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, p. 37. Springer, Heidelberg (1998)

[Peled et al. 1999] Peled, D., Vardi, M.Y., Yannakakis, M.: Black-box Checking. In: Wu, J., et al. (eds.) Formal Methods for Protocol Engineering and Distributed Systems, FORTE/PSTV, Beijing, pp. 225–240. Kluwer, Dordrecht (1999)

[Porat, Feldman 1991] Porat, S., Feldman, J.: Learning automata from ordered examples. Machine Learning 7, 109–138 (1991)

[Raffelt et al. 2008] Raffelt, H., Steffen, B., Margaria, T.: Dynamic Testing Via Automata Learning. In: Yorav, K. (ed.) HVC 2007. LNCS, vol. 4899, pp. 136–152. Springer, Heidelberg (2008)

Appendix 1: Case Study

The automaton to be CGE learned has state set = { 0, 1, 2, 3, 4 }
Input alphabet = { 0, 1 } , Output alphabet = { 0, 1 } , Starting state = 0
Transition/Output Table =

	0	1	2	3	4
0	2/0	0/0	3/1	0/0	4/1
1	0/0	0/0	1/0	4/1	3/1

1-st observation = (0) : [] 0,
R = { (0) → () , (1) → () } , Lambda_R = { (0) → 0 }

2-nd observation = (0, 0, 0) : [0, 1] 0 : Run CGE - inconsistent
R = { (1) → (), (0, 0) → (), (0, 1) → () }
Lambda_R = { (0) → 0, (0, 0) → 1 }

3-rd observation = (0, 0, 0, 0) : [0, 1, 0] 0 : Run CGE - inconsistent
R = { (1) → (), (0, 1) → (), (0, 0, 0) → (), (0, 0, 1) → () }
Lambda_R = { (0) → 0, (1) → 0, (0, 0) → 1, (0, 0, 0) → 0 }

4-th observation = (0, 1, 0, 0) : [0, 0, 0] 0 : Run CGE - inconsistent
R = { (1) → (), (0, 1) → (0, 0), (0, 0, 0) → (), (0, 0, 1) → () }
Lambda_R = { (0) → 0, (1) → 0, (0, 0) → 1, (0, 1) → 0, (0, 0, 0) → 0 }

```
5-th observation = (0, 0, 1, 0) : [0, 1, 1] 1 : Run CGE - inconsistent
R = { (1) → (), (0, 1) → (0, 0), (0, 0, 0) → (), (0, 0, 1) → (0,) }
Lambda_R= { (0) → 0, (1) → 0, (0, 0) → 1, (0, 1) → 0, (0, 0, 0) → 0, (0, 0, 1) → 1 }
──────────────
6-th observation = (0, 1, 1) : [0, 0] 0 : Run CGE - inconsistent
R = { (1) → (), (0, 0, 0) → (), (0, 0, 1) → (0), (0, 1, 0) → (), (0, 1, 1) → () }
Lambda_R = { (0) → 0, (1) → 0, (0, 0) → 1, (0, 1) → 0, (0, 0, 0) → 0, (0, 0, 1) →
1, (0, 1, 0) → 0, (0, 1, 1) → 0 }
──────────────
7-th observation = (0, 0, 1, 0, 0, 0) : [0, 1, 1, 1, 1] 1 : Run CGE - inconsistent
R = { (1) → (), (0, 0, 0) → (), (0, 1, 0) → (), (0, 1, 1) → (), (0, 0, 1, 0) → (0, 0, 1),
(0, 0, 1, 1) → () }
Lambda_R = { (0) → 0, (1) → 0, (0, 0) → 1, (0, 1) → 0, (0, 0, 0) → 0, (0, 0, 1) →
1, (0, 1, 0) → 0, (0, 1, 1) → 0, (0, 0, 1, 0) → 1 }
──────────────
8-th observation = (0, 0, 1, 1, 0, 0) : [0, 1, 1, 1, 0] 0 : Run CGE - inconsistent
R = { (1) → (), (0, 0, 0) → (), (0, 1, 0) → (), (0, 1, 1) → (), (0, 0, 1, 0) → (0, 0, 1),
(0, 0, 1, 1) → (0, 0) }
Lambda_R = { (0) → 0, (1) → 0, (0, 0) → 1, (0, 1) → 0, (0, 0, 0) → 0, (0, 0, 1) →
1, (0, 1, 0) → 0, (0, 1, 1) → 0, (0, 0, 1, 0) → 1, (0, 0, 1, 1) → 1 }
────────────────────
Learning complete, computation time = 0.15s, average time per observation = 0.01871s
```