Learning Weighted Automata
over Principal Ideal Domains⋆

Gerco van Heerdt[1], Clemens Kupke[2](✉), Jurriaan Rot[1,3], and Alexandra Silva[1]

[1] University College London, United Kingdom
{gerco.heerdt,alexandra.silva}@ucl.ac.uk
[2] University of Strathclyde, United Kingdom
clemens.kupke@strath.ac.uk
[3] Radboud University, The Netherlands
jrot@cs.ru.nl

Abstract. In this paper, we study active learning algorithms for weighted automata over a semiring. We show that a variant of Angluin's seminal L⋆ algorithm works when the semiring is a principal ideal domain, but not for general semirings such as the natural numbers.

1 Introduction

Angluin's seminal L⋆ algorithm [4] for active learning of deterministic automata (DFAs) has been successfully used in many verification tasks, including in automatically building formal models of chips in bank cards or finding bugs in network protocols (see [27,14] for a broad overview of successful applications of active learning). While DFAs are expressive enough to capture interesting properties, certain verification tasks require more expressive models. This motivated several researchers to extend L⋆ to other types of automata, notably Mealy machines [28,24], register automata [15,22,1], and nominal automata [20].

Weighted finite automata (WFAs) are an important model made popular due to their applicability in image processing and speech recognition tasks [11,21]. The model is prevalent in other areas, including bioinformatics [2] and formal verification [3]. Passive learning algorithms and associated complexity results have appeared in the literature (see e.g. [5] for an overview), whereas active learning has been less studied [6,7]. Furthermore, the existing learning algorithms, both passive and active, have been developed assuming the weights in the automaton are drawn from a field, such as the real numbers.[4] To the best of our knowledge, no learning algorithms, whether passive or active, have been developed for WFAs in which the weights are drawn from a general semiring.

[4] Balle and Mohri [6] define WFAs generically over a semiring but then restrict to fields from Section 3 onwards as they present an overview of existing learning algorithms.

In this paper, we explore *active learning* for WFAs over a general semiring. The main contributions of the paper are as follows:

1. We introduce a weighted variant of $L^\star$ parametric on an arbitrary semiring, together with sufficient conditions for termination (Section 4).
2. We show that for general semirings our algorithm might not terminate. In particular, if the semiring is the natural numbers, one of the steps of the algorithm might not converge (Section 5).
3. We prove that the algorithm terminates if the semiring is a *principal ideal domain*, covering the known case of fields, but also the integers. This yields the first active learning algorithm for WFAs over the integers (Section 6).

We start in Section 2 by explaining the learning algorithm for WFAs over the reals and pointing out the challenges in extending it to arbitrary semirings.

2   Overview of the Approach

In this section, we give an overview of the work developed in the paper through examples. We start by informally explaining the general algorithm for learning weighted automata that we introduce in Section 4, for the case where the semiring is a field. More specifically, for simplicity we consider the field of real numbers throughout this section. Later in the section, we illustrate why this algorithm does not work for an arbitrary semiring.

Angluin's $L^\star$ algorithm provides a procedure to learn the minimal DFA accepting a certain (unknown) regular language. In the weighted variant we will introduce in Section 4, for the specific case of the field of real numbers, the algorithm produces the minimal WFA accepting a weighted rational language (or formal power series) $\mathcal{L} \colon A^* \to \mathbb{R}$.

A WFA over $\mathbb{R}$ consists of a set of states, a linear combination of initial states, a transition function that for each state and input symbol produces a linear combination of successor states, and an output value in $\mathbb{R}$ for each state (Definition 5). As an example, consider the WFA over $A = \{a\}$ below.
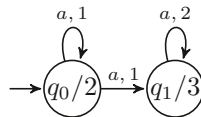
Here q_0 is the only initial state, with weight 1, as indicated by the arrow into it that has no origin. When reading a, q_0 transitions with weight 1 to itself and also with weight 1 to q_1; q_1 transitions with weight 2 just to itself. The output of q_0 is 2 and the output of q_1 is 3.

The language of a WFA is determined by letting it read a given word and determining the final output according to the weights and outputs assigned to individual states. More precisely, suppose we want to read the word aaa in the example WFA above. Initially, q_0 is assigned weight 1 and q_1 weight 0. Processing the first a then leads to q_0 retaining weight 1, as it has a self-loop with weight 1,

and q_1 obtaining weight 1 as well. With the next a, the weight of q_0 still remains 1, but the weight of q_1 doubles due to its self-loop of weight 1 and is added to the weight 1 coming from q_0, leading to a total of 3. Similarly, after the last a the weights are 1 for q_0 and 7 for q_1. Since q_0 has output 2 and q_1 output 3, the final result is $2 \cdot 1 + 3 \cdot 7 = 23$.

The learning algorithm assumes access to a *teacher* (sometimes also called *oracle*), who answers two types of queries:

- *membership queries*, consisting of a single word $w \in A^*$, to which the teacher replies with a weight $\mathcal{L}(w) \in \mathbb{R}$;
- *equivalence queries*, consisting of a hypothesis WFA $\mathcal{A}$, to which the teacher replies **yes** if its language $\mathcal{L}_{\mathcal{A}}$ equals the target language $\mathcal{L}$ and **no** otherwise, providing a counterexample $w \in A^*$ such that $\mathcal{L}(w) \neq \mathcal{L}_{\mathcal{A}}(w)$.

In practice, membership queries are often easily implemented by interacting with the system one wants to model the behaviour of. However, equivalence queries are more complicated—as the perfect teacher does not exist and the target automaton is not known they are commonly approximated by testing. Such testing can however be done exhaustively if a bound on the number of states of the target automaton is known. Equivalence queries can also be implemented exactly when learning algorithms are being compared experimentally on generated automata whose languages form the targets. In this case, standard methods for language equivalence, such as the ones based on bisimulations [9], can be used.

The learning algorithm incrementally builds an *observation table*, which at each stage contains partial information about the language $\mathcal{L}$ determined by two finite sets $S, E \subseteq A^*$. The algorithm fills the table through membership queries. As an example, and to set notation, consider the following table (over $A = \{a\}$).

$$
\begin{array}{c|ccc}
 & \varepsilon & a & aa \\
\hline
\varepsilon & 0 & 1 & 3 \\
a & 1 & 3 & 7 \\
aa & 3 & 7 & 15
\end{array}
$$

$$
\begin{aligned}
&\mathsf{row} \colon S \to \mathbb{R}^E \\
&\mathsf{row}(u)(v) = \mathcal{L}(uv) \\[4pt]
&\mathsf{srow} \colon S \cdot A \to \mathbb{R}^E \\
&\mathsf{srow}(ua)(v) = \mathcal{L}(uav)
\end{aligned}
$$

This table indicates that $\mathcal{L}$ assigns 0 to ε, 1 to a, 3 to aa, 7 to aaa, and 15 to $aaaa$. For instance, we see that $\mathsf{row}(a)(aa) = \mathsf{srow}(aa)(a) = 7$. Since row and srow are fully determined by the language $\mathcal{L}$, we will refer to an observation table as a pair (S, E), leaving the language $\mathcal{L}$ implicit.

If the observation table (S, E) satisfies certain properties described below, then it represents a WFA (S, δ, i, o), called the *hypothesis*, as follows:

- $\delta \colon S \to (\mathbb{R}^S)^A$ is a linear map defined by choosing for $\delta(s)(a)$ a linear combination over S of which the rows evaluate to $\mathsf{srow}(sa)$;
- $i \colon S \to \mathbb{R}$ is the initial weight map defined as $i(\varepsilon) = 1$ and $i(s) = 0$ for $s \neq \varepsilon$;
- $o \colon S \to \mathbb{R}$ is the output weight map defined as $o(s) = \mathsf{row}(s)(\varepsilon)$.

For this to be well-defined, we need to have $\varepsilon \in S$ (for the initial weights) and $\varepsilon \in E$ (for the output weights), and for the transition function there is a crucial property of the table that needs to hold: closedness. In the weighted setting, a table is closed if for all $t \in S \cdot A$, there exist $r_s \in \mathbb{R}$ for all $s \in S$ such that

$$\mathsf{srow}(t) = \sum_{s \in S} r_s \cdot \mathsf{row}(s).$$

If this is not the case for a given $t \in S \cdot A$, the algorithm adds t to S. The table is repeatedly extended in this manner until it is closed. The algorithm then constructs a hypothesis, using the closedness witnesses to determine transitions, and poses an equivalence query to the teacher. It terminates when the answer is **yes**; otherwise it extends the table with the counterexample provided by adding all its suffixes to E, and the procedure continues by closing again the resulting table. In the next subsection we describe the algorithm through an example.

Remark 1. The original $\mathsf{L}^\star$ algorithm requires a second property to construct a hypothesis, called *consistency*. Consistency is difficult to check in extended settings, so the present paper is based on a variant of the algorithm inspired by Maler and Pnueli [19] where only closedness is checked and counterexamples are handled differently. See [13] for an overview of consistency in different settings.

2.1   Example: Learning a Weighted Language over the Reals

Throughout this section we consider the following weighted language:

$$\mathcal{L} \colon \{a\}^* \to \mathbb{R} \qquad\qquad \mathcal{L}(a^j) = 2^j - 1.$$

The minimal WFA recognising it has 2 states. We will illustrate how the weighted variant of Angluin's algorithm recovers this WFA.

We start from $S = E = \{\varepsilon\}$, and fill the entries of the table on the left below by asking membership queries for ε and a. The table is not closed and hence we build the table on its right, adding the membership result for aa. The resulting table is closed, as $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a)$, so we construct the hypothesis $\mathcal{A}_1$.

	ε
ε	0
a	1

	ε
ε	0
a	1
aa	3

$$\mathcal{A}_1 = \quad \rightarrow \boxed{q_0/0} \xrightarrow{a,\,1} \boxed{q_1/1} \circlearrowright a,\,3$$

$$q_0 = \varepsilon$$
$$q_1 = a$$

The teacher replies **no** and gives the counterexample aaa, which is assigned 9 by the hypothesis automaton $\mathcal{A}_1$ but 7 in the language. Therefore, we extend $E \leftarrow E \cup \{a, aa, aaa\}$. The table becomes the one below. It is closed, as $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a) - 2 \cdot \mathsf{row}(\varepsilon)$, so we construct a new hypothesis $\mathcal{A}_2$.

	ε	a	aa	aaa
ε	0	1	3	7
a	1	3	7	15
aa	3	7	15	31
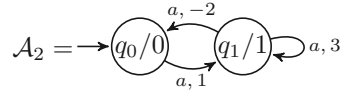
The teacher replies **yes** because $\mathcal{A}_2$ accepts the intended language assigning $2^j - 1 \in \mathbb{R}$ to the word a^j, and the algorithm terminates with the correct automaton.

2.2   Learning Weighted Languages over Arbitrary Semirings

Consider now the same language as above, but represented as a map over the semiring of natural numbers $\mathcal{L}\colon \{a\}^* \to \mathbb{N}$ instead of a map $\mathcal{L}\colon \{a\}^* \to \mathbb{R}$ over the reals. Accordingly, we consider a variant of the learning algorithm over the semiring $\mathbb{N}$ rather than the algorithm over $\mathbb{R}$ described above. For the first part, the run of the algorithm for $\mathbb{N}$ is the same as above, but after receiving the counterexample we can no longer observe that $\mathsf{srow}(aa) = 3 \cdot \mathsf{row}(a) - 2 \cdot \mathsf{row}(\varepsilon)$, since $-2 \notin \mathbb{N}$. In fact, there are no $m, n \in \mathbb{N}$ such that $\mathsf{srow}(aa) = m \cdot \mathsf{row}(\varepsilon) + n \cdot \mathsf{row}(a)$. To see this, consider the first two columns in the table and note that $\frac{3}{7}$ is bigger than $\frac{0}{1} = 0$ and $\frac{1}{3}$, so it cannot be obtained as a linear combination of the latter two using natural numbers. We thus have a closedness defect and update $S \leftarrow S \cup \{aa\}$, leading to the table below.

	ε	a	aa	aaa
ε	0	1	3	7
a	1	3	7	15
aa	3	7	15	31
aaa	7	15	31	63

Again, the table is not closed, since $\frac{7}{15} > \frac{3}{7}$. In fact, these closedness defects continue appearing indefinitely, leading to non-termination of the algorithm. This is shown formally in Section 5.

Note, however, that there does exist a WFA over $\mathbb{N}$ accepting this language:

(1)

The reason that the algorithm cannot find the correct automaton is closely related to the algebraic structure induced by the semiring. In the case of the reals, the algebras are vector spaces and the closedness checks induce increases in the dimension of the hypothesis WFA, which in turn cannot exceed the dimension of the minimal one for the language. In the case of commutative monoids, the algebras for the natural numbers, the notion of dimension does not exist and unfortunately the algorithm does not terminate. In Section 6 we show that one can get around this problem for a class of semirings which includes the integers.

We mentioned earlier that during experimental evaluation the target WFA is known, and equivalence queries may be implemented via standard language equivalence methods. A further issue with arbitrary semirings is that language equivalence can be undecidable; that is the case, e.g., for the tropical semiring.

In Section 3 we recall basic definitions used throughout the paper, after which Section 4 introduces our general algorithm with its (parameterised) termination proof of Theorem 14. We then proceed to prove non-termination of the example discussed above over the natural numbers in Section 5 before instantiating our algorithm to PIDs in Section 6 and showing that it terminates in Theorem 28. We conclude with a discussion of related and future work in Section 7.

3   Preliminaries

Throughout this paper we fix a semiring[5] $\mathbb{S}$ and a finite alphabet A. We start with basic definitions related to semimodules and weighted languages.

Definition 2 (Semimodule). *A* (left) *semimodule M over $\mathbb{S}$ consists of a monoid structure on M, written using $+$ as the operation and 0 as the unit, together with a scalar multiplication map $\cdot\colon \mathbb{S} \times M \to M$ such that:*

$$s \cdot 0_M = 0_M \qquad\qquad 0_{\mathbb{S}} \cdot m = 0_M \qquad\qquad 1 \cdot m = m$$
$$s \cdot (m + n) = s \cdot m + s \cdot n \quad (s + r) \cdot m = s \cdot m + r \cdot m \quad (sr) \cdot m = s \cdot (r \cdot m).$$

When the semiring is in fact a ring, we speak of a module *rather than a semimodule. In the case of a field, the concept instantiates to a vector space.*

As an example, commutative monoids are the semimodules over the semiring of natural numbers. Any semiring forms a semimodule over itself by instantiating the scalar multiplication map to the internal multiplication. If X is any set and M is a semimodule, then M^X with pointwise operations also forms a semimodule. A similar semimodule is the *free semimodule* over X, which differs from M^X in that it fixes M to be $\mathbb{S}$ and requires its elements to have *finite support*. This enables an important operation called *linearisation*.

Definition 3 (Free semimodule). *The* free semimodule *over a set X is given by the set*
$$V(X) = \{f\colon X \to \mathbb{S} \mid \operatorname{supp}(f) \text{ is finite}\}$$
with pointwise operations. Here $\operatorname{supp}(f) = \{x \in X \mid f(x) \neq 0\}$. We sometimes identify the elements of $V(X)$ with formal sums over X. Any semimodule isomorphic to $V(X)$ for some set X is called free.

If X is a finite set, then $V(X) = \mathbb{S}^X$. We now define *linearisation* of a function into a semimodule, which uniquely extends it to a semimodule homomorphism, witnessing the fact that $V(X)$ is free.

---

[5] Rings and semirings considered in this paper are taken to be unital.

Definition 4 (Linearisation). *Given a set X, a semimodule M, and a function $f\colon X \to M$, we define the* linearisation *of f as the semimodule homomorphism $f^{\sharp}\colon V(X) \to M$ given by*

$$f^{\sharp}(\alpha) = \sum_{x \in X} \alpha(x) \cdot f(x).$$

The $(-)^{\sharp}$ operation has an inverse that maps a semimodule homomorphism $g\colon V(X) \to M$ to the function $g^{\dagger}\colon X \to M$ given by

$$g^{\dagger}(x) = g(\partial_x), \qquad\qquad \partial_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{if } y \neq x. \end{cases}$$

We proceed with the definition of WFAs and their languages.

Definition 5 (WFA). *A* weighted finite automaton *(WFA) over $\mathbb{S}$ is a tuple (Q, δ, i, o), where Q is a finite set, $\delta\colon Q \to (\mathbb{S}^Q)^A$, and $i, o\colon Q \to \mathbb{S}$.*

A *weighted language* (or just *language*) over $\mathbb{S}$ is a function $A^* \to \mathbb{S}$. To define the language accepted by a WFA $\mathcal{A} = (Q, \delta, i, o)$, we first introduce the notions of *observability map* $\mathsf{obs}_{\mathcal{A}}\colon V(Q) \to \mathbb{S}^{A^*}$ and *reachability map* $\mathsf{reach}_{\mathcal{A}}\colon V(A^*) \to V(Q)$ as the semimodule homomorphisms given by

$$\mathsf{reach}^{\dagger}_{\mathcal{A}}(\varepsilon) = i \qquad\qquad \mathsf{obs}_{\mathcal{A}}(m)(\varepsilon) = o^{\sharp}(m)$$
$$\mathsf{reach}^{\dagger}_{\mathcal{A}}(ua) = \delta^{\sharp}(\mathsf{reach}^{\dagger}_{\mathcal{A}}(u))(a) \qquad \mathsf{obs}_{\mathcal{A}}(m)(au) = \mathsf{obs}_{\mathcal{A}}(\delta^{\sharp}(m)(a))(u).$$

The *language accepted by a WFA* $\mathcal{A} = (Q, \delta, i, o)$ is the function $\mathcal{L}_{\mathcal{A}}\colon A^* \to \mathbb{S}$ given by $\mathcal{L}_{\mathcal{A}} = \mathsf{obs}_{\mathcal{A}}(i)$. Equivalently, one can define this as $\mathcal{L}_{\mathcal{A}} = o^{\sharp} \circ \mathsf{reach}^{\dagger}_{\mathcal{A}}$.

4   General Algorithm for WFAs

In this section we define the general algorithm for WFAs over $\mathbb{S}$, as described informally in Section 2. Our algorithm assumes the existence of a *closedness strategy* (Definition 8), which allows one to check whether a table is closed, and in case it is, provide relevant witnesses. We then introduce sufficient conditions on $\mathbb{S}$ and on the language $\mathcal{L}$ to be learned under which the algorithm terminates.

Definition 6 (Observation table). *An* observation table *(or just* table*) (S, E) consists of two sets $S, E \subseteq A^*$. We write $\mathsf{Table_{fin}} = \mathcal{P}_f(A^*) \times \mathcal{P}_f(A^*)$ for the set of finite tables (where $\mathcal{P}_f(X)$ denotes the collection of finite subsets of a set X). Given a language $\mathcal{L}\colon A^* \to \mathbb{S}$, an observation table (S, E) determines the* row function $\mathsf{row}_{(S,E,\mathcal{L})}\colon S \to \mathbb{S}^E$ *and the* successor row *function $\mathsf{srow}_{(S,E,\mathcal{L})}\colon S \cdot A \to \mathbb{S}^E$ as follows:*

$$\mathsf{row}_{(S,E,\mathcal{L})}(w)(v) = \mathcal{L}(wv) \qquad \mathsf{srow}_{(S,E,\mathcal{L})}(wa)(v) = \mathcal{L}(wav).$$

We often write $\mathsf{row}_{\mathcal{L}}$ and $\mathsf{srow}_{\mathcal{L}}$, or even row and srow, when the parameters are clear from the context.

A table is *closed* if the successor rows are linear combinations of the existing rows in S. To make this precise, we use the linearisation $\mathsf{row}^\sharp$ (Definition 4), which extends row to linear combinations of words in S.

Definition 7 (Closedness). *Given a language $\mathcal{L}$, a table (S, E) is* closed *if for all $w \in S$ and $a \in A$ there exists $\alpha \in V(S)$ such that $\mathsf{srow}(wa) = \mathsf{row}^\sharp(\alpha)$.*

This corresponds to the notion of closedness described in Section 2.

A further important ingredient of the algorithm is a method for checking whether a table is closed. This is captured by the notion of closedness strategy.

Definition 8 (Closedness strategy). *Given a language $\mathcal{L}$, a* closedness strategy *for $\mathcal{L}$ is a family of computable functions*

$$\big(\mathsf{cs}_{(S,E)} \colon S \cdot A \to \{\bot\} \cup V(S)\big)_{(S,E) \in \mathsf{Table}_{\mathsf{fin}}}$$

satisfying the following two properties:

- *if $\mathsf{cs}_{(S,E)}(t) = \bot$, then there is no $\alpha \in V(S)$ s.t. $\mathsf{row}^\sharp(\alpha) = \mathsf{srow}(t)$, and*
- *if $\mathsf{cs}_{(S,E)}(t) \neq \bot$, then $\mathsf{row}^\sharp(\mathsf{cs}_{(S,E)}(t)) = \mathsf{srow}(t)$.*

Thus, given a closedness strategy as above, a table (S, E) is closed iff $\mathsf{cs}_{(S,E)}(t) \neq \bot$ for all $t \in S \cdot A$. More specifically, for each $t \in S \cdot A$ we have that $\mathsf{cs}_{(S,E)}(t) \neq \bot$ iff the (successor) row corresponding to t already forms a linear combination of rows labelled by S. In that case, this linear combination is returned by $\mathsf{cs}_{(S,E)}(t)$. This is used to close tables in our learning algorithm, introduced below.

Examples of semirings and (classes of) languages that admit a closedness strategy are described at the end of this section. Important for our algorithm will be that closedness strategies are computable. This problem is equivalent to solving systems of equations $A\underline{x} = \underline{b}$, where A is the matrix whose columns are $\mathsf{row}(s)$ for $s \in S$, $\underline{x}$ is a vector of length $|S|$, and $\underline{b}$ is the vector consisting of the row entries in $\mathsf{srow}(t)$ for some $t \in S \cdot A$. These observations motivate the following definition.

Definition 9 (Solvability). *A semiring $\mathbb{S}$ is* solvable *if a solution to any finite system of linear equations of the form $A\underline{x} = \underline{b}$ is computable.*

We have the following correspondence.

Proposition 10. *For any language accepted by a WFA over any semiring there exists a closedness strategy if and only if the semiring is solvable.*

Proof. If the semiring is solvable, we obtain a closedness strategy by the remarks prior to Definition 9. Conversely, we can construct a language that is non-zero on finitely many words and encode in a table (S, E) a given linear equation. To be able to freely choose the value in each table cell, we can consider a sufficiently large alphabet to make sure S and E contain only single-letter words. This avoids dependencies within the table. $\square$

Algorithm 1 Abstract learning algorithm for WFA over $\mathbb{S}$

1: $S, E \leftarrow \{\varepsilon\}$
2: **while** true **do**
3:     **while** $\mathsf{cs}_{(S,E)}(t) = \bot$ for some $t \in S \cdot A$ **do**
4:         $S \leftarrow S \cup \{t\}$
5:     **for** $s \in S$ **do**
6:         $o(s) \leftarrow \mathsf{row}_{\mathcal{L}}(s)(\varepsilon)$
7:         **for** $a \in A$ **do**
8:             $\delta(s)(a) \leftarrow \mathsf{cs}_{(S,E)}(sa)$
9:     **if** $\mathsf{EQ}(S, \delta, \varepsilon, o) = w \in A^*$ **then**
10:         $E \leftarrow E \cup \mathsf{suffixes}(w)$
11:     **else**
12:         **return** $(S, \delta, \varepsilon, o)$

We now have all the ingredients to formulate the algorithm to learn weighted languages over a general semiring. The pseudocode is displayed in Algorithm 1.

The algorithm keeps a table (S, E), and starts by initialising both S and E to contain just the empty word. The inner while loop (lines 3–4) uses the closedness strategy to repeatedly check whether the current table is closed and add new rows in case it is not. Once the table is closed, a hypothesis is constructed, again using the closedness strategy (lines 5–8). This hypothesis $(S, \delta, \varepsilon, o)$ is then given to the teacher for an equivalence check. The equivalence check is modelled by EQ (line 9) as follows: if the hypothesis is incorrect, the teacher non-deterministically returns a counterexample $w \in A^*$, the condition evaluates to true, and the suffixes of w are added to E; otherwise, if the hypothesis is correct, the condition on line 9 evaluates to false, and the algorithm returns the correct hypothesis on line 12.

4.1 Termination of the General Algorithm

The main question remaining is: under which conditions does this algorithm terminate and hence learns the unknown weighted language? We proceed to give abstract conditions under which it terminates. There are two main assumptions:

1. A way of measuring progress the algorithm makes with the observation table when it distinguishes linear combinations of rows that were previously equal, together with a bound on this progress (Definition 11).
2. An assumption on the *Hankel matrix* of the input language (Definition 12), which makes sure we encounter finitely many closedness defects throughout any run of the algorithm. More specifically, we assume that the Hankel matrix satisfies a finite approximation property (Definition 13).

The first assumption is captured by the definition of progress measure:

Definition 11 (Progress measure). *A* progress measure *for a language $\mathcal{L}$ is a function* $\mathsf{size} \colon \mathsf{Table}_{\mathsf{fin}} \to \mathbb{N}$ *such that*

(a) *there exists $n \in \mathbb{N}$ such for all $(S, E) \in \mathsf{Table}_{\mathsf{fin}}$ we have $\mathsf{size}(S, E) \leq n$;*

(b) *given $(S, E), (S, E') \in \mathsf{Table}_{\mathsf{fin}}$ and $s_1, s_2 \in V(S)$ such that $E \subseteq E'$ and $\mathsf{row}^{\sharp}_{(S,E,\mathcal{L})}(s_1) = \mathsf{row}^{\sharp}_{(S,E,\mathcal{L})}(s_2)$ but $\mathsf{row}^{\sharp}_{(S,E',\mathcal{L})}(s_1) \neq \mathsf{row}^{\sharp}_{(S,E',\mathcal{L})}(s_2)$, we have $\mathsf{size}(S, E') > \mathsf{size}(S, E)$.*

A progress measure assigns a 'size' to each table, in such a way that (a) there is a global bound on the size of tables, and (b) if we extend a table with some proper tests in E, i.e., such that some combinations of rows in $\mathsf{row}^{\sharp}$ that were equal before get distinguished by a newly added test, then the size of the extended table is properly above the size of the original table. This is used to ensure that, when adding certain counterexamples supplied by the teacher, the size of the table, measured according to the above size function, properly increases.

The second assumption that we use for termination is phrased in terms of the Hankel matrix associated to the input language $\mathcal{L}$, which represents $\mathcal{L}$ as the (semimodule generated by the) infinite table where both the rows and columns contain all words. The Hankel matrix is defined as follows.

Definition 12 (Hankel matrix). *Given a language $\mathcal{L} \colon A^* \to \mathbb{S}$, the semimodule generated by a table (S, E) is given by the image of $\mathsf{row}^{\sharp}$. We refer to the semimodule generated by the table (A^*, A^*) as the Hankel matrix of $\mathcal{L}$.*

The Hankel matrix is approximated by the tables that occur during the execution of the algorithm. For termination, we will therefore assume that this matrix satisfies the following finite approximation condition.

Definition 13 (Ascending chain condition). *We say that a semimodule M satisfies the ascending chain condition if for all inclusion chains of subsemimodules of M,*

$$S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots ,$$

there exists $n \in \mathbb{N}$ such that for all $m \geq n$ we have $S_m = S_n$.

Given the notions of progress measure, Hankel matrix and ascending chain condition, we can formulate the general theorem for termination of Algorithm 1.

Theorem 14 (Termination of the abstract learning algorithm). *In the presence of a progress measure, Algorithm 1 terminates whenever the Hankel matrix of the target language satisfies the ascending chain condition (Definition 13).*

Proof. Suppose the algorithm does not terminate. Then there is a sequence $\{(S_n, E_n)\}_{n \in \mathbb{N}}$ of tables where (S_0, E_0) is the initial table and (S_{n+1}, E_{n+1}) is formed from (S_n, E_n) after resolving a closedness defect or adding columns due to a counterexample.

We write H_n for the semimodule generated by the table (S_n, A^*). We have $S_n \subseteq S_{n+1}$ and thus $H_n \subseteq H_{n+1}$. Note that a closedness defect for (S_n, E_n) is also a closedness defect for (S_n, A^*), so if we resolve the defect in the next step, the inclusion $H_n \subseteq H_{n+1}$ is strict. Since these are all included in the Hankel matrix, which satisfies the ascending chain condition, there must be an n such that for all $k \geq n$ we have that (S_k, E_k) is closed.

In [13, Section 6] it is shown that in a general table used for learning automata with side-effects given by a monad there exists a suffix of each counterexample for the corresponding hypothesis that when added as a column label leads to either a closedness defect or to distinguishing two combinations of rows in the table. Since WFAs are automata with side-effects given by the free semimodule monad[6] and we add all suffixes of the counterexample to the set of column labels, this also happens in our algorithm. Thus, for all $k \geq n$ where we process a counterexample, there must be two linear combinations of rows distinguished, as closedness is already guaranteed. Then the semimodule generated by (S_k, E_k) is a strict quotient of the semimodule generated by (S_{k+1}, E_{k+1}). By the progress measure we then find $\mathsf{size}(S_k, E_k) < \mathsf{size}(S_{k+1}, E_{k+1})$, which cannot happen infinitely often. We conclude that the algorithm must terminate.   □

To illustrate the hypotheses needed for Algorithm 1 and its termination (Theorem 14), we consider two classes of semirings for which learning algorithms are already known in the literature [7,13].

Example 15 (Weighted languages over fields). Consider any field for which the basic operations are computable. Solvability is then satisfied via a procedure such as Gaussian elimination, so by Proposition 10 there exists a closedness strategy. Hence, we can instantiate Algorithm 1 with $\mathbb{S}$ being such a field.

For termination, we show that the hypotheses of Theorem 14 are satisfied whenever the input language is accepted by a WFA. First, a progress measure is given by the dimension of the vector space generated by the table. To see this, note that if we distinguish two linear combinations of rows, we can assume without loss of generality that one of these linear combinations in the extended table uses only basis elements. This in turn can be rewritten to distinguishing a single row from a linear combination of rows using field operations, with the property that the extended version of the single row is a basis element. Hence, the row was not a basis element in the original table, and therefore the dimension of the vector space generated by the table has increased. Adding rows and columns cannot decrease this dimension, so it is bounded by the dimension of the Hankel matrix. Since the language we want to learn is accepted by a WFA, the associated Hankel matrix has a finite dimension [10,12] (see also, e.g., [5]), providing a bound for our progress measure.

Finally, for any ascending chain of subspaces of the Hankel matrix, these subspaces are of finite dimension bounded by the dimension of the Hankel matrix. The dimension increases along a strict subspace relation, so the chain converges.

Example 16 (Weighted languages over finite semirings). Consider any finite semiring. Finiteness allows us to apply a brute force approach to solving systems of equations. This means the semiring is solvable, and hence a closedness strategy exists by Proposition 10.

For termination, we can define a progress measure by assigning to each table the size of the image of $\mathsf{row}^\sharp$. Distinguishing two linear combinations of rows

---

[6] We note that [13] assumes the monad to preserve finite sets. However, the relevant arguments do not depend on this.

increases this measure. If the language we want to learn is accepted by a WFA, then the Hankel matrix contains a subset of the linear combinations of the languages of its states. Since there are only finitely many such linear combinations, the Hankel matrix is finite, which bounds our measure. A finite semimodule such as the Hankel matrix in this case does not admit infinite chains of subspaces. We conclude by Theorem 14 that Algorithm 1 terminates for the instance that the semiring $\mathbb{S}$ is a finite, if the input language is accepted by a WFA over $\mathbb{S}$.

For the Boolean semiring, an instance of the above finite semiring example, WFAs are non-deterministic finite automata. The algorithm we recover by instantiating Algorithm 1 to this case is close to the algorithm first described by Bollig et al. [8]. The main differences are that in their case the hypothesis has a state space given by a minimally generating subset of the distinct rows in the table rather than all elements of S, and they do apply a notion of consistency.

In Section 6 we will show that Algorithm 1 can learn WFAs over principal ideal domains—notably including the integers—thus providing a strict generalisation of existing techniques.

5   Issues with Arbitrary Semirings

We concluded the previous section with examples of semirings for which Algorithm 1 terminates if the target language is accepted by a WFA. In this section, we prove a negative result for the algorithm over the semiring $\mathbb{N}$: we show that it does not terminate on a certain language over $\mathbb{N}$ accepted by a WFA over $\mathbb{N}$, as anticipated in Section 2.2. This means that Algorithm 1 does not work well for arbitrary semirings. The problem is that the Hankel matrix of a language recognised by WFA does not necessarily satisfy the ascending chain condition that is used to prove Theorem 14. In the example given in the proof below, the Hankel matrix is not even finitely generated.

Theorem 17. *There exists a WFA $\mathcal{A}_{\mathbb{N}}$ over $\mathbb{N}$ such that Algorithm 1 does not terminate when given $\mathcal{L}_{\mathcal{A}_{\mathbb{N}}}$ as input, regardless of the closedness strategy used.*

Proof. Let $\mathcal{A}_{\mathbb{N}}$ be the automaton over the alphabet $\{a\}$ given in (1) in Section 2.2. Formally, $\mathcal{A}_{\mathbb{N}} = (Q, \delta, i, o)$, where

$$Q = \{q_0, q_1\} \qquad i = q_0 \qquad o(q_0) = 0$$
$$\delta(q_0)(a) = q_0 + q_1 \qquad \delta(q_1)(a) = 2q_1 \qquad o(q_1) = 1.$$

As mentioned in Section 2.2, the language $\mathcal{L} \colon \{a\}^* \to \mathbb{N}$ accepted by $\mathcal{A}_{\mathbb{N}}$ is given by $\mathcal{L}(a^j) = 2^j - 1$. This can be shown more precisely as follows. First one shows by induction on j that $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_1)(a^j) = 2^j$ for all $j \in \mathbb{N}$—we leave the straightforward argument to the reader. Second, we show, again by induction on j, that $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^j) = 2^j - 1$. This implies the claim, as $\mathcal{L} = \mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)$. For $j = 0$ we have $\mathsf{obs}_{\mathcal{A}_{\mathbb{N}}}(q_0)(a^j) = o(q_0) = 0 = 2^0 - 1$ as required. For the inductive

step, let $j = k+1$ and assume $\mathsf{obs}_{\mathcal{A}_\mathbb{N}}(q_0)(a^k) = 2^k - 1$. We calculate

$$\begin{aligned}
\mathsf{obs}_{\mathcal{A}_\mathbb{N}}(q_0)(a^{k+1}) &= \mathsf{obs}_{\mathcal{A}_\mathbb{N}}(q_0 + q_1)(a^k) \\
&= \mathsf{obs}_{\mathcal{A}_\mathbb{N}}(q_0)(a^k) + \mathsf{obs}_{\mathcal{A}_\mathbb{N}}(q_1)(a^k) \\
&= (2^k - 1) + 2^k \\
&= 2^{k+1} - 1.
\end{aligned}$$

Note that in particular the language $\mathcal{L}$ is injective.

Towards a contradiction, suppose the algorithm does terminate with table (S, E). Let $J = \{j \in \mathbb{N} \mid a^j \in S\}$ and define $n = \max(J)$. Since the algorithm terminates with table (S, E), the latter must be closed. In particular, there exist $k_j \in \mathbb{N}$ for all $j \in J$ such that $\sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j) = \mathsf{srow}_{\mathcal{L}}(a^n a)$. We consider two cases. First assume $E = \{\varepsilon\}$ and let $\mathcal{A} = (Q', \delta', i', o')$ be the hypothesis. For all $l \in \mathbb{N}$ we have $\mathsf{row}_{\mathcal{L}}^\sharp(\mathsf{reach}_{\mathcal{A}}^\dagger(a^l))(\varepsilon) = 2^l - 1$ because $\mathcal{A}$ must be correct. Thus, if $a^l \in S \cdot A$, then $\mathsf{row}_{\mathcal{L}}^\sharp(\mathsf{reach}_{\mathcal{A}}^\dagger(a^l)) = \mathsf{srow}_{\mathcal{L}}(a^l)$. In particular,

$$\mathsf{row}_{\mathcal{L}}^\sharp(\mathsf{reach}_{\mathcal{A}}^\dagger(a^n a)) = \mathsf{srow}_{\mathcal{L}}(a^n a) = \sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j).$$

Note that we can choose the k_j such that $\mathsf{reach}_{\mathcal{A}}^\dagger(a^n a) = \sum_{j \in J} k_j \cdot a^j$. Since

$$\begin{aligned}
\mathsf{row}_{\mathcal{L}}^\sharp\left(\delta'^\sharp\left(\sum_{j \in J} k_j \cdot a^j\right)(a)\right) &= \mathsf{row}_{\mathcal{L}}^\sharp\left(\sum_{j \in J} k_j \cdot \delta'(a^j)(a)\right) \\
&= \sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(\delta'(a^j)(a)) \\
&= \sum_{j \in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a),
\end{aligned}$$

we have $\mathsf{row}_{\mathcal{L}}^\sharp(\mathsf{reach}_{\mathcal{A}}^\dagger(a^n aa)) = \sum_{j \in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)$ and therefore

$$\sum_{j \in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)(\varepsilon) = \mathsf{row}_{\mathcal{L}}^\sharp(\mathsf{reach}_{\mathcal{A}}^\dagger(a^n aa))(\varepsilon) = 2^{n+2} - 1.$$

Then

$$\begin{aligned}
2^{n+2} - 1 &= \sum_{j \in J} k_j \cdot \mathsf{srow}_{\mathcal{L}}(a^j a)(\varepsilon) = \sum_{j \in J} k_j (2^{j+1} - 1) \\
&= 2\left(\sum_{j \in J} k_j (2^j - 1)\right) + \sum_{j \in J} k_j = 2(2^{n+1} - 1) + \sum_{j \in J} k_j,
\end{aligned}$$

so $\sum_{j \in J} k_j = 1$. This is only possible if there is $j_1 \in J$ s.t. $k_{j_1} = 1$ and $k_j = 0$ for all $j \in J \setminus \{j_1\}$. However, this implies that $\mathsf{row}_{\mathcal{L}}(a^{j_1}) = \mathsf{srow}_{\mathcal{L}}(a^n a)$, which contradicts injectivity of $\mathcal{L}$ as $n \geq j_1$. Thus, the algorithm did not terminate.

For the other case, assume there is $a^m \in E$ such that $m \geq 1$. We have

$$2^{n+1} - 1 = \mathsf{srow}_{\mathcal{L}}(a^n a)(\varepsilon) = \sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j)(\varepsilon) = \sum_{j \in J} k_j(2^j - 1),$$

so

$$\sum_{j \in J} k_j(2^{j+m} - 1) = \sum_{j \in J} k_j \cdot \mathsf{row}_{\mathcal{L}}(a^j)(a^m)$$
$$= \mathsf{srow}_{\mathcal{L}}(a^n a)(a^m)$$
$$= 2^{n+m+1} - 1$$
$$= 2^m(2^{n+1} - 1) + 2^m - 1$$
$$= 2^m \left(\sum_{j \in J} k_j(2^j - 1) \right) + 2^m - 1$$
$$= \left(\sum_{j \in J} k_j(2^{j+m} - 2^m) \right) + 2^m - 1$$
$$= \left(\sum_{j \in J} k_j(2^{j+m} - 1) \right) + \left(\sum_{j \in J} k_j(1 - 2^m) \right) + 2^m - 1.$$

Then

$$\left(\sum_{j \in J} k_j(1 - 2^m) \right) + 2^m - 1 = 0.$$

Since $m \geq 1$ this is only possible if there is $j_1 \in J$ s.t. $k_{j_1} = 1$ and $k_j = 0$ for all $j \in J \setminus \{j_1\}$. However, this implies $\mathsf{row}_{\mathcal{L}}(a^{j_1}) = \mathsf{srow}_{\mathcal{L}}(a^n a)$, which again contradicts injectivity of $\mathcal{L}$ as $n \geq j_1$. Thus, the algorithm did not terminate. $\quad\square$

Remark 18. Our proof shows non-termination for a bigger class of algorithms than Algorithm 1; it uses only the definition of the hypothesis, that closedness is satisfied before constructing the hypothesis, that S and E contain the empty word, and that termination implies correctness. For instance, adding the prefixes of a counterexample to S instead of its suffixes to E will not fix the issue.

We have thus shown that our algorithm does not instantiate to a terminating one for an arbitrary semiring. To contrast this negative result, in the next section we identify a class of semirings not previously explored in the learning literature where we can guarantee a terminating instantiation.

6   Learning WFAs over PIDs

We show that for a subclass of semirings, namely *principal ideal domains (PIDs)*, the abstract learning algorithm of Section 4 terminates. This subclass includes

the integers, Gaussian integers, and rings of polynomials in one variable with coefficients in a field. We will prove that the Hankel matrix of a language over a PID accepted by a WFA has analogous properties to those of vector spaces—finite rank, a notion of progress measure, and the ascending chain condition. We also give a sufficient condition for PIDs to be solvable, which by Proposition 10 guarantees the existence of a closedness strategy for the learning algorithm.

To define PIDs, we first need to introduce ideals. Given a ring $\mathbb{S}$, a *(left) ideal* I of $\mathbb{S}$ is an additive subgroup of $\mathbb{S}$ s.t. for all $s \in \mathbb{S}$ and $i \in I$ we have $si \in I$. The ideal I is *(left) principal* if it is of the form $I = \mathbb{S}s$ for some $s \in \mathbb{S}$.

Definition 19 (PID). *A principal ideal domain $\mathbb{P}$ is a non-zero commutative ring in which every ideal is principal and where for all $p_1, p_2 \in \mathbb{P}$ such that $p_1 p_2 = 0$ we have $p_1 = 0$ or $p_2 = 0$.*

A module M over a PID $\mathbb{P}$ is called *torsion free* if for all $p \in \mathbb{P}$ and any $m \in M$ such that $p \cdot m = 0$ we have $p = 0$ or $m = 0$. It is a standard result that a module over a PID is torsion free if and only if it is free [17, Theorem 3.10].

The next definition of *rank* is analogous to that of the dimension of a vector space and will form the basis for the progress measure.

Definition 20 (Rank). *We define the* rank *of a finitely generated free module $V(X)$ over a PID as* $\mathsf{rank}(V(X)) = |X|$.

This definition extends to any finitely generated free module over a PID, as $V(X) \cong V(Y)$ for finite sets X and Y implies $|X| = |Y|$ [17, Theorem 3.4].

Now that we have a candidate for a progress measure function, we need to prove it has the required properties. The following lemmas will help with this.

Lemma 21. *Given finitely generated free modules M, N over a PID s.t. $\mathsf{rank}(M) \geq \mathsf{rank}(N)$, any surjective module homomorphism $f \colon N \to M$ is injective.*

Proof. Since $\mathsf{rank}(M) \geq \mathsf{rank}(N)$, there exists a surjective module homomorphism $g \colon M \to N$. Therefore $g \circ f \colon N \to N$ is surjective and by [23] an iso. In particular, f is injective. $\square$

Lemma 22. *If M and N are finitely generated free modules over a PID such that there exists a surjective module homomorphism $f \colon N \to M$, then $\mathsf{rank}(M) \leq \mathsf{rank}(N)$. If f is not injective, then $\mathsf{rank}(M) < \mathsf{rank}(N)$.*

Proof. Let $f \colon N \to M$ be a surjective module homomorphism. Suppose towards a contradiction that $\mathsf{rank}(M) > \mathsf{rank}(N)$. By Lemma 21 f is injective, so M is isomorphic to a submodule of N and $\mathsf{rank}(M) \leq \mathsf{rank}(N)$ [17]; contradiction.

For the second part, suppose f is not injective and assume towards a contradiction that $\mathsf{rank}(M) \geq \mathsf{rank}(N)$. Again by Lemma 21 f is injective, which is a contradiction with our assumption. Thus, in this case $\mathsf{rank}(M) < \mathsf{rank}(N)$. $\square$

The lemma below states that the Hankel matrix of a weighted language over a PID has finite rank which bounds the rank of any module generated by an observation table. This will be used to define a progress measure, used to prove termination of the learning algorithm for weighted languages over PIDs.

Lemma 23 (Hankel matrix rank for PIDs). *When targeting a language accepted by a WFA over a PID, any module generated by an observation table is free. Moreover, the Hankel matrix has finite rank that bounds the rank of any module generated by an observation table.*

Proof. Given a WFA $\mathcal{A} = (Q, \delta, i, o)$, let M be the free module generated by Q. Note that the Hankel matrix is the image of the composition $\mathsf{obs}_{\mathcal{A}} \circ \mathsf{reach}_{\mathcal{A}}$. Consider the image of the module homomorphism $\mathsf{reach}_{\mathcal{A}} \colon V(A^*) \to M$, which we write as R. Since R is a submodule of M, we know from [17] that R is free and finitely generated with $\mathsf{rank}(R) \leq \mathsf{rank}(M)$. The Hankel matrix can now be obtained as the image of the restriction of $\mathsf{obs}_{\mathcal{A}} \colon M \to \mathbb{S}^{A^*}$ to the domain R. Let H be this image, which we know is finitely generated because R is. Since H is a submodule of the torsion free module $\mathbb{S}^{A^*}$, it is also torsion free and therefore free. We also have a surjective module homomorphism $s \colon R \to H$, so by Lemma 22 we find $\mathsf{rank}(H) \leq \mathsf{rank}(R)$.

Let N be the module generated by an observation table (S, E). We have that N is a quotient of the module generated by (S, A^*), which in turn is a submodule of H. Using again [17] and Lemma 22 we conclude that N is free and finitely generated with $\mathsf{rank}(N) \leq \mathsf{rank}(H)$.   □

The second part of Lemma 23 would follow from a PID variant of Fliess' theorem [12]. We are not aware of such a result, and leave this for future work.

Proposition 24 (Progress measure for PIDs). *There exists a progress measure for any language accepted by a WFA over a PID.*

Proof. Define $\mathsf{size}(S, E) = \mathsf{rank}(M)$, where M is the module generated by the table (S, E). By Lemma 23 this is bounded by the rank of the Hankel matrix. If M and N are modules generated by two tables such that N is a strict quotient of M, then by Lemma 22 we have $\mathsf{rank}(M) > \mathsf{rank}(N)$.   □

Recall that, for termination of the algorithm, Theorem 14 requires a progress measure, which we defined above, and it requires the Hankel matrix of the language to satisfy the ascending chain condition (Definition 13). Proposition 25 shows that the latter is always the case for languages over PIDs.

Proposition 25 (Ascending chain condition PIDs). *The Hankel matrix of a language accepted by a WFA over a PID satisfies the ascending chain condition.*

Proof. Let H be the Hankel matrix, which has finite rank by Lemma 23. If

$$M_1 \subseteq M_2 \subseteq M_3 \subseteq \cdots$$

is any chain of submodules of H, then $M = \bigcup_{i \in \mathbb{N}} M_i$ is a submodule of H and therefore also of finite rank [17]. Let B be a finite basis of M. There exists $n \in \mathbb{N}$ such that $B \subseteq M_n$, so $M_n = M$.   □

The last ingredient for the abstract algorithm is solvability of the semiring: the following fact provides a sufficient condition for a PID to be solvable.

Proposition 26 (PID solvability). *A PID $\mathbb{P}$ is solvable if all of its ring operations are computable and if each element of $\mathbb{P}$ can be effectively factorised into irreducible elements.*

Proof. It is well-known that a system of equations of the form $A\underline{x} = \underline{b}$ with integer coefficients can be efficiently solved via computing the Smith normal form [25] of A. The algorithm generalises to principal ideal domains, if we assume that the factorisation of any given element of the principal ideal domain[7] into irreducible elements is computable, cf. the algorithm in [16, p. 79-84]. To see that all steps in this algorithm can be computed, one has to keep in mind that the factorisation can be used to determine the greatest common divisor of any two elements of the principal ideal domain.   □

Remark 27. In the case that we are dealing with an Euclidean domain $\mathbb{P}$, a sufficient condition for $\mathbb{P}$ to be solvable is that Euclidean division is computable (again this can be deduced from inspecting the algorithm in [16, p. 79-84]). Such a PID behaves essentially like the ring of integers.

Putting everything together, we obtain the main result of this section.

Theorem 28 (Termination for PIDs). *Algorithm 1 can be instantiated and terminates for any language accepted by a WFA over a PID of which all ring operations are computable and of which each element can be effectively factorised into irreducible elements.*

Proof. To instantiate the algorithm, we need a closedness strategy. According to Proposition 10 it is sufficient for the PID to be solvable, which is shown by Proposition 26. Proposition 24 provides a progress measure, and we know from Proposition 25 that the Hankel matrix satisfies the ascending chain condition, so by Theorem 14 the algorithm terminates.   □

The example run given in Section 2.1 is the same when performed over the integers. We note that if the teacher holds an automaton model of the correct language, equivalence queries are decidable by lifting the embedding of the PID into its *quotient field* to the level of WFAs and checking equivalence there.

7   Discussion

We have introduced a general algorithm for learning WFAs over arbitrary semirings, together with sufficient conditions for termination. We have shown an inherent termination issue over the natural numbers and proved termination for PIDs. Our work extends the results by Bergadano and Varricchio [7], who showed that WFAs over fields could be learned from a teacher. Although we note that a PID can be embedded into its corresponding field of fractions, the WFAs produced when learning over the field potentially have weights outside the PID.

---

[7] Note that factorisations exist as each principal ideal domain is also a unique factorisation domain, cf. e.g. [17, Thm. 2.23].

Algorithmic issues with WFAs over arbitrary semirings have been identified before. For instance, Krob [18] showed that language equivalence is undecidable for WFAs over the tropical semiring.

On the technical level, a variation on WFAs is given by probabilistic automata, where transitions point to convex rather than linear combinations of states. One easily adapts the example from Section 5 to show that learning probabilistic automata has a similar termination issue. On the positive side, Tappler et al. [26] have shown that deterministic MDPs can be learned using an $L^\star$ based algorithm. The deterministic MDPs in *loc.cit.* are very different from the automata in our paper, as their states generate observable output that allows to identify the current state based on the generated input-output sequence.

One drawback of the ascending chain condition on the Hankel matrix is that this does not give any indication of the number of steps the algorithm requires. Indeed, the submodule chains traversed, although converging, may be arbitrarily long. We would like to measure and bound the progress made when fixing closedness defects, but this turns out to be challenging for PIDs. The rank of the module generated by the table may not increase. We leave an investigation of alternative measures to future work.

We would also like to adapt the algorithm so that for PIDs it always produces minimal automata. At the moment this is already the case for fields,[8] since adding a row due to a closedness defect preserves linear independence of the image of row. For PIDs things are more complicated—adding rows towards closedness may break linear independence and thus a basis needs to be found in row$^\sharp$. This complicates the construction of the hypothesis.

Our results show that, on the one hand, WFAs can be learned over finite semirings and arbitrary PIDs (assuming computability of the relevant operations) and, on the other hand, that there exists an infinite commutative semiring for which they cannot be learned. However, there are many classes of semirings in between commutative semirings and PIDs, of which we would like to know whether their WFAs can be learned by our general algorithm.

Finally, we would like to generalise our results to extend the framework introduced in [13], which focusses on learning automata with side-effects over a monad. WFAs as considered in the present paper are an instance of those, where the monad is the free semimodule monad $V(-)$. At the moment, the results in [13] apply to a monad that preserves finite sets, but much of our general WFA learning algorithm and termination argument can be extended to that setting. It would be interesting to see if crucial properties of PIDs that lead to a progress measure and to satisfying the ascending chain condition could also be translated to the monad level.

---

[8] There is one exception: the language that assigns 0 to every word, which is accepted by a WFA with no states. The algorithm initialises the set of row labels, which constitute the state space of the hypothesis, with the empty word.

References

1. Fides Aarts, Paul Fiterau-Brostean, Harco Kuppens, and Frits W. Vaandrager. Learning register automata with fresh value generation. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *ICTAC*, volume 9399 of *LNCS*, pages 165–183. Springer, 2015.

2. Cyril Allauzen, Mehryar Mohri, and Ameet Talwalkar. Sequence kernels for predicting protein essentiality. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 9–16. ACM, 2008.

3. Benjamin Aminof, Orna Kupferman, and Robby Lampert. Formal analysis of online algorithms. In Tevfik Bultan and Pao-Ann Hsiung, editors, *ATVA*, volume 6996 of *LNCS*, pages 213–227. Springer, 2011.

4. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

5. Borja Balle and Mehryar Mohri. Spectral learning of general weighted automata via constrained matrix completion. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 2168–2176, 2012.

6. Borja Balle and Mehryar Mohri. Learning weighted automata. In Andreas Maletti, editor, *CAI*, volume 9270 of *LNCS*, pages 1–21. Springer, 2015.

7. Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, December 1996.

8. Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In Craig Boutilier, editor, *IJCAI*, pages 1004–1009, 2009.

9. Michele Boreale. Weighted bisimulation in linear algebraic form. In *CONCUR*, volume 5710 of *LNCS*, pages 163–177. Springer, 2009.

10. Jack W. Carlyle and Azaria Paz. Realizations by stochastic finite automata. *J. Comput. Syst. Sci.*, 5(1):26–40, 1971.

11. Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1993.

12. Michel Fliess. Matrices de Hankel. *J. Math. Pures Appl*, 53(9):197–222, 1974.

13. Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. Optimizing automata learning via monads. *arXiv preprint arXiv:1704.08055*, 2017.

14. Falk Howar and Bernhard Steffen. Active automata learning in practice - an annotated bibliography of the years 2011 to 2016. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172*, volume 11026 of *LNCS*, pages 123–148. Springer, 2018.

15. Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV*, volume 9206 of *LNCS*, pages 487–495. Springer, 2015.

16. Nathan Jacobson. *Lectures in Abstract Algebra*, volume 31 of *GTM*. Springer, 1953.

17. Nathan Jacobson. *Basic algebra I*. Courier Corporation, 2012.

18. Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

19. Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inform. and Comput.*, 118:316–326, 1995.
20. Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *POPL*, pages 613–625. ACM, 2017.
21. Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. *CoRR*, abs/cs/0503077, 2005.
22. Malte Mues, Falk Howar, Kasper Søe Luckow, Temesghen Kahsai, and Zvonimir Rakamaric. Releasing the PSYCO: using symbolic search in interface generation for java. *ACM SIGSOFT Software Engineering Notes*, 41(6):1–5, 2016.
23. Morris Orzech. Onto endomorphisms are isomorphisms. *The American Mathematical Monthly*, 78(4):357–362, 1971.
24. Muzammil Shahbaz and Roland Groz. Inferring Mealy machines. In *FM*, volume 5850 of *LNCS*, pages 207–222, Berlin, Heidelberg, 2009. Springer-Verlag.
25. Henry J. Stephen Smith. On systems of linear indeterminate equations and congruences. *Philosophical Transactions of the Royal Society of London*, 151:293–326, 1861.
26. Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-Based Learning of Markov Decision Processes. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *FM*, volume 11800 of *LNCS*, pages 651–669. Springer, 2019.
27. Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.
28. Juan Miguel Vilar. Query learning of subsequential transducers. In *ICGI*, volume 1147 of *LNCS*, pages 72–83. Springer, 1996.