Seminar Algorithmic Learning Theory, SS 2015
Prof. Dr. Peter Rossmanith, Dr. Somnath Sikdar

Pattern Languages

Michael Krause

RWTH Aachen University, Aachen, Germany,
michael.krause@rwth-aachen.de.

May 14, 2015

Contents

1 Introduction

This report deals with a type of formal languages called *Pattern Languages*, which were introduced by Dana Angluin in 1980. More specifically, the report discusses the results from Angluin's original paper [1], an important algorithm presented by Steffen Lange and Rolf Wiehagen [9] and some results from various other papers.

Angluin's investigation of pattern languages was originally motivated by the results of E. Mark Gold on language identification in the limit [6]. In his seminal work, Gold proposes a model of *inductive inference*, which is the problem of deriving general rules from specific examples. This seems to be an essential part of what we understand as "learning something". In theoretical computer science, we are primarily interested in learning formal languages.

Gold's model describes learning of formal languages as a process: at each time step a learner is being presented with information on the language, whereupon he must derive hypotheses about which language is being presented. If at some point, the learner arrives at a correct hypothesis and does not change it afterwards, the language is considered learned. In this framework, the learner may be presented either with strings from the language (positive data) or any kind of strings including an indication on whether they belong to the language or not (positive and negative data). Gold showed, that there are many classes of formal languages that can be learned from positive and negative data, but not from positive data only. [6]

This raises the question, under which circumstances a language can be learned from positive data alone. Angluin found conditions for this in [2]. Furthermore, she presented the pattern languages in [1] and showed that they are inferable from positive data. The pattern language are of particular interest, because they work as a formalization of the problem of inductive inference.

A pattern is any finite string made up of constants and variables. For instance, $1x01y0x$ is a pattern where x, y are variables and $0, 1$ are constants. From this, other patterns can be constructed by substituting finite strings for the variables, while the constants remain the same. Thus, with different substitutions the above pattern produces the strings $1101001, 10y010100y, 1001x00$ and many more.

The main problem discussed in [1] is that of finding a pattern descriptive of a set of strings of constants. Intuitively, the question is which pattern generates every string from the set so that no other pattern produces a more precise answer. For example, when given the set of strings $\{101, 000\}$, the pattern x produces every string in the set, but the pattern $x0x$ is more precise. Since this problem is decidable but NP-complete, answers have been sought for special cases or different variations of pattern languages.

We will begin by introducing some general ideas and notation, whereupon we will closely look at Angluin's paper, Lange and Wiehagen's algorithm and mention some other findings that have been made since.

2 Basic ideas

This section introduces the formal definition of patterns and related concepts as in [1].

Let Σ be a finite alphabet of *constants* containing at least two characters and X a set of *variables* disjoint from Σ. A *pattern* is any finite string from $(X \cup \Sigma)^+$. P_k is the set of all patterns containing k variables. Then $P_* = \bigcup_{k=0}^{\infty} P_k$ is the *set of all patterns*.

A *substitution* is a mapping $h : P_* \to P_*$, which is the identity when restricted to Σ, so that for all patterns p, q: $h(p) \neq \epsilon$ and $h(p \cdot q) = h(p) \cdot h(q)$. This formalizes the intuitive notion of substituting non-null strings for variables in a pattern, so that a variable is always replaced with the same string and constants remain unchanged. A *renaming* of variables is a substitution which is also bijective on X.

Let p, q be two patterns. We call p *less general than* q (i.e: $p \leqslant' q$) if and only if there exists a substitution h so that $p = h(q)$, e.g: $1010011 \leqslant' xx01y$. We say q *generates* p if and only if $p \leqslant' q$ and $p \in \Sigma^+$. The *language of* p is defined as the set of all strings of constants that p generates, so $L(p) = \{s \in \Sigma^+ : s \leqslant' p\}$

We call p and q *equivalent* (i.e: $p \equiv' q$) if and only if there exists a renaming r so that $p = r(q)$, e.g: $yy01x \equiv' xx01y$. A pattern is in *canonical form* if its variables are $\{x_1, x_2, \ldots\}$ and x_i appears first before the first occurrence of x_{i+1} for all i. Therefore, if $p = z10yxx$ and $q = x_1 10 x_2 x_3 x_3$, then q is in canonical form and $p \equiv' q$

We call any finite, nonempty $S \subset \Sigma^+$ a *sample*. We say a pattern p generates a sample S if and only if it generates every $s \in S$, i.e. $S \subseteq L(p)$. The pattern $p = x$ generates every sample. We call a pattern p *descriptive* of S if and only if p generates S and for every $q \neq p$ that also generates S, $L(q)$ is not a proper subset of $L(p)$.

Recall that we described inductive inference as the problem of finding general rules for examples. We may now formalize the problem like this: Given a sample S, which pattern is descriptive of S?

3 Finding Patterns Common to a Set of Strings

At first glance, the problem of finding descriptive patterns may seem quite easy to solve. Let S be a pattern. Then an intuitive attempt at solving the problem might be:

1. Enumerate all patterns of shorter or equal length of the shortest string in S
2. Test for each pattern if its language contains S
3. From all patterns that pass the test: Select one whose language is minimal with regards to inclusion

Clearly, the above steps result in a descriptive pattern. In step 1, we can omit patterns of a certain length, because in our definition of substitution, empty substitutions are not allowed. Thus, a pattern can not generate strings shorter than itself. Step 2 requires that we can check, whether a given string is generated by a pattern. Step 3 requires that we can test, whether one pattern language is included in another.

In order to analyse whether the aforementioned procedure is indeed effective, we therefore have to look at some properties of pattern languages first.

3.1 Properties of pattern languages

We will look at some common decision problems for formal languages and their solutions for pattern languages. Here, p, q denote patterns and $s \in \Sigma^+$ a string of constants. Unless noted otherwise, all results are from [1].

Table 1 compares the properties of pattern languages to those of other formal languages. While pattern languages might appear similar to regular or context-free languages, they are in fact different from both. The copy language $L(xx)$ is not context-free and thus not regular. However, even some regular languages can not be described by a pattern. For example, the regular language $L(0 \mid 1) = \{0, 1\}$ is not a pattern language, because a pattern language always contains either just one element (if the pattern contains no variables) or infinitely many elements (otherwise). Nevertheless, all pattern languages are context-sensitive (theorem 20.4 in [7]).

Problem 1 (Equivalence)**.** $L(p) \stackrel{?}{=} L(q)$

Note, that the corresponding problem for context-free languages is undecidable. For pattern languages, however, it is easy to solve because of the following theorem:

Theorem (3.5 in [1])**.** *For all patterns p,q:* $L(p) = L(q) \iff p \equiv' q$

It is easy to verify whether $p \equiv' q$ by constructing the patterns' canonical forms and checking, if they are the same. A pattern's canonical form can be constructed in linear time by simply iterating through it from beginning to end and replacing the variable names with $x_1, x_2, \ldots$ in ascending order. Thus, the equivalence problem for pattern languages is decidable in time linear in the length of the input patterns.

Problem 2 (Membership)**.** $s \stackrel{?}{\in} L(p)$

The corresponding problem for context-free languages is decidable in P. For pattern languages, it is in NP, as one could guess a substitution for p and verify in polynomial time, whether it yields s. A reduction to the 3SAT decision problem then gives us:

Theorem (3.6 in [1])**.** *The membership problem for pattern languages is* NP*-complete*

Problem 3 (Inclusion)**.** $L(p) \stackrel{?}{\subseteq} L(q)$

This problem remained unsolved in [1]. However, it was shown there that for the special cases where either p and q have the same length or q is a pattern with only one variable: $L(p) \subseteq L(q) \iff p \leqslant' q$. The decision problem whether $p \leqslant' q$ for two patterns p, q is decidable (by guessing a substitution) but NP-complete, as it could be used to solve the membership problem described above.

For the general case, the question was answered in [8]:

Theorem (5.1 in [8])**.** *The inclusion problem for arbitrary pattern languages is undecidable*

Language	Membership	Emptiness	Equivalence	Inclusion
Context-sensitive	D	U	U	U
Context-free	D	D	U	U
Regular	D	D	D	D
Pattern languages	D	D	D	U

Table 1: Properties of formal languages. D=decidable, U=undecidable

3.2 Finding descriptive patterns

The results summarized in the previous section show that our initial attempt at finding descriptive patterns does not work: The inclusion test in step 3 is undecidable. Because of this, Angluin proposes a slightly different approach in [1]:

1. Enumerate all patterns of shorter or equal length of the shortest string in S
2. Test for each pattern if its language contains S
3. From all patterns that pass the test: select the longest
4. From the resulting set of patterns, output any which is minimal with regards to $\leqslant'$

As described in the previous section, step 4 is equivalent to finding patterns whose language is minimal with regards to inclusion. Because of step 3, this procedure will only find descriptive patterns of maximum length. Additionally, the amount of patterns found in step 1 grows exponentially with the length of the shortest string in S. Step 2 requires a membership test for each string in S, which is NP-complete. The same applies for the tests required to find $\leqslant'$-minimal patterns in step 4. [1]

These problems are not unique to the procedure defined above. As Angluin shows in the proof to the following theorem, if there was a polynomial-time algorithm to find descriptive patterns of maximum possible length, the membership problem for pattern languages would be in P:

Theorem (4.2 in [1])**.** *If* $\mathsf{P} \neq \mathsf{NP}$ *then there is no polynomial-time algorithm to find a pattern of maximum possible length descriptive of* S

Despite this theorem, we may still find special cases, where descriptive patterns can be found in polynomial time. One such case will be discussed in the next section.

3.3 Finding descriptive one-variable patterns

A main result of [1] is a polynomial-time algorithm for finding descriptive patterns with only one variable. This section describes the algorithm while leaving out some of the more technical considerations in the original paper. We will start by giving a general overview of how the algorithm works.

The algorithm partitions the set of all one-variable patterns into patterns that share a common structure, described by a triple. Only patterns that have triples which are

feasible for all input strings are considered. For each feasible triple, an automaton is constructed that recognizes exactly those patterns which have this triple and generate every string in the input sample. From the resulting set of automata, a specific subset is selected and the resulting automata are those, that recognize patterns descriptive of the sample.

We will now introduce the notion of *feasible triples*. The function $\tau : P_1 \to \mathbb{N}^3$ assigns each one-variable pattern p a triple (i, j, k) where i is the number of constants, j the number of occurrences of the variable x and k the position of the first occurrence of x in p. Let s be a string of constants. Then a triple (i, j, k) is called *feasible* for s if it satisfies the following conditions:

1. $0 \leq j \leq |s|$
2. $0 \leq i < |s|$
3. $1 \leq k \leq i + 1$
4. $j \mid (|s| - i)$

A pattern p can only generate s, if $\tau(p)$ is feasible for s. Accordingly, p may only generate a sample S, if $\tau(p)$ is feasible for every $s \in S$. This makes conditions 1 and 3 plausible. Condition 2 guarantees, that we have at least one variable in the pattern (patterns without any variable can only be descriptive of the set containing just themselves). Condition 4 gives us the length of the substring t that is substituted for x in p: $|t| = (|s| - i)/j$

Let F be the set of all triples feasible for S. It is shown in [1], that $|F| \in \mathcal{O}(l^2 \log l)$ (where l is the smallest length of a string in S). Therefore, we can construct F in polynomial time.

Now, assume we have a feasible triple (i, j, k) for a string of constants s. We construct an automaton $B_s(i, j, k)$ that processes patterns and accepts only those that generate s and satisfy $\tau(p) = (i, j, k)$. Its states are tuples (c, v) with semantics similar to the aforementioned triples, so c corresponds to the number of constants and v to the number of x read so far. The initial state is $(0, 0)$ and the only final state is (i, j).

Let t be the substring of s as defined by the triple, i.e: t begins at position k of s and has length $(|s| - i)/j$. This gives us for every state (c, v) of our automaton a position $w = c + v * |t|$ in the string s. In other words: if our automaton reaches state (c, v), then the pattern read so far accounts for w characters in s.

Based on the characters in s, we create transitions for every state (c, v):

- $(c, v) \xrightarrow{b} (c + 1, v)$, if $(c, v) \neq (k - 1, 0)$ and the symbol at position $w + 1$ of s is $b \in \Sigma$. In other words: we create constant transitions for every character in s except at the position of the first occurrence of x.
- $(c, v) \xrightarrow{x} (c, v + 1)$, if $(c, v) \neq (u, 0)$ (for $u < k - 1$) and t occurs in s at position $w + 1$. Therefore: we create variable transitions whenever the substring t occurs in s, but not before the position of the first occurrence of x.

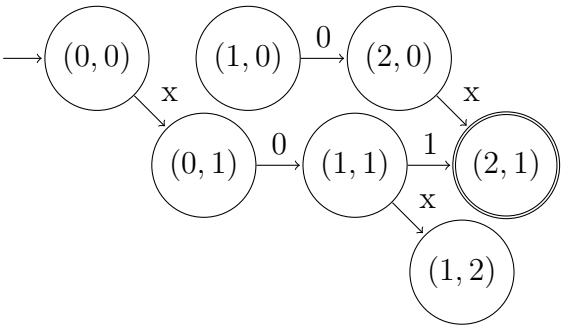

Figure 1: Example automaton $B_{101}(2, 1, 1)$

Example. *Given* $s = 101$ *and the feasible triple* $(2, 1, 1)$ *for* s*, we get the substring* $t = 1$ *and* $B_s(2, 1, 1)$ *as seen in figure 1.*

The language of $B_s(i, j, k)$ is exactly those one-variable patterns p with $\tau(p) = (i, j, k)$ that generate s. Now, assume we have an input sample S. We want to get an automaton $B(i, j, k)$ that recognizes one-variable patterns p with $\tau(p) = (i, j, k)$ which generate *every* string in S. For this, we must construct $B_s(i, j, k)$ for every $s \in S$ and then intersect all of them. The intersection automaton $B(i, j, k)$ retains from these automata only those states, transitions and final states that existed in *every* automaton it was constructed from.

With this construction we get:

Lemma (6.2 in [1])**.** $\bigcup_{(i,j,k)\in F} L(B(i, j, k))$ *is exactly the set of one-variable patterns that generate every string in* S

We can now describe Angluin's algorithm for finding descriptive one-variable patterns for a sample S:

1. Construct F by enumerating all feasible triples
2. For each $(i, j, k) \in F$ construct $B(i, j, k)$
3. Select (i', j', k') with $L(B(i', j', k')) \neq \emptyset$ and $i' + j'$ maximum

Then any pattern $p \in L(B(i', j', k'))$ is descriptive of S among one variable patterns. In step 3, we discard automata with empty languages, because this means that there are no patterns with the corresponding triple that generate every string in S. The decision to maximize $i' + j'$ can be justified as follows:

Assume we found p through the algorithm and p was *not* descriptive of S among one-variable patterns. Then there is some one-variable pattern q that generates S and $L(q) \subsetneq L(p)$. As mentioned in section 3.1, $L(q) \subseteq L(p) \iff q \leqslant' p$. So since there must be a substitution h from p to q, we get that $|q| \geq |p|$, because a substitution never produces strings shorter than the original pattern. As p maximizes $i + j$ among one-variable patterns that generate S, we get $|p| \geq |q|$ and thus $|p| = |q|$. Therefore, either h substitutes a constant into p (so that q is made up of constants) or h replaces

the variable x in p with another variable. In the latter case, p and q are equivalent and so $L(q) = L(p)$ (cf. section 3.1). In both cases we get a contradiction to the assumption that $L(q) \subsetneq L(p)$. All in all, p is descriptive of S.

As mentioned before, we can bound the number of feasible triples. Furthermore, we construct the automata in time polynomial in their sizes, which are bound by the length of the input strings. Thus, the algorithm runs in time polynomial in the length of the input.

Example. *We will illustrate Angluin's algorithm through an example. Let* $S = \{s_1, s_2, s_3\}$ *a sample with* $s_1 = 1101011, s_2 = 10011, s_3 = 11111$*. We want to find a one-variable pattern descriptive of* S*.*

Firstly, we construct F *by enumerating and intersecting the sets of feasible triples for* s_1, s_2 *and* s_3*. We get:* $F = \{(1,1,k), (1,2,k), (2,1,k), (3,1,k), (3,2,k), (4,1,k)\}$ *for* $1 \leq k \leq i+1$ *in each of the triples.*

Secondly, we construct $B(i,j,k)$ *for each triple in* F*. In this example, we do the construction for* $(3,2,2) \in F$*: we construct* $B_s(3,2,2)$ *for every* $s \in S$ *(see appendix A) and then intersect the three automata (see figure 2).*

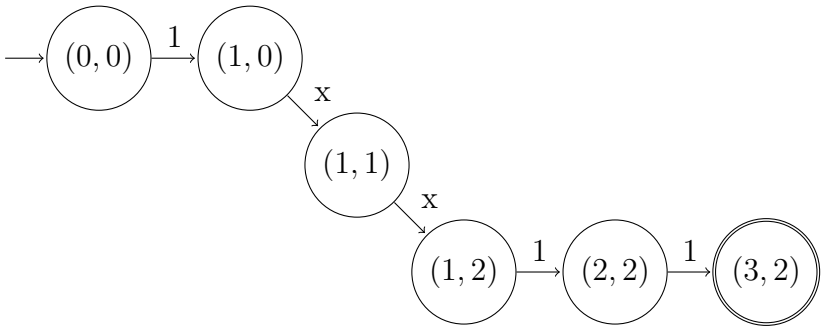

Figure 2: Intersection automaton $B(3,2,2)$ (reduced)

The language recognized by the intersection automaton is $L(B(3,2,2)) = \{1xx11\} \neq \emptyset$*. Since* $3+2$ *is maximum for* $i+j$ *in* F*,* $1xx11$ *is descriptive of* S *among one-variable patterns.*

4 Polynomial-time Inference of Arbitrary Pattern Languages

4.1 Learning pattern languages in the limit

We begin by putting the problem of finding descriptive patterns into the context of Gold's learning model of learning in the limit [6]. Here, the objects being learned are formal languages that are presented as a sequence of strings coming from the language so that each string appears at least once (in Gold's words: a text). The learner outputs

hypotheses after receiving each string. We say the learner learns the language, if, after some finite amount of time, the hypotheses are correct and remain the same.

In our case, we assume the learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language $L(p)$. The learner's hypotheses are patterns descriptive of the strings seen so far. Angluin proves in [2] (theorem 5) that there is a point in time, when the learner's hypothesis will be the pattern p and consequently, it will not be changed after receiving other strings from $L(p)$. This way, pattern languages can be learned in the limit from positive data.

As discussed before, we know of a polynomial-time algorithm to find descriptive one-variable patterns, but nothing equivalent for patterns with an arbitrary number of variables. We call the time needed to compute a new hypothesis the *update time* of a learner. Without a polynomial-time algorithm to compute descriptive patterns, the update time seems to be without polynomial bound.

This section presents an algorithm introduced by Steffen Lange and Rolf Wiehagen in [9]. The algorithm learns any pattern language in the limit and has polynomial update time. However, it is *inconsistent*, meaning that it will sometimes output wrong hypotheses (i.e. patterns that do not generate every string seen so far).

Before we introduce the algorithm, we describe the key concept of [9], which is finding descriptive patterns of the same length.

4.2 Finding descriptive patterns of the same length

Let $p = p_1 \ldots p_m$ a pattern and s a string of the same length. We construct a pattern $q = q_1 \ldots q_m$ for each $1 \leq i \leq m$ as follows:

$$q_i = \begin{cases} p_i & \text{if } p_i = s_i \quad (1) \\ x & \text{if } p_i \neq s_i \text{ and there is a position } k < i, \text{ where} \quad (2) \\ & s_k = s_i \text{ and } p_k = p_i \text{ and } q_k = x \\ x & \text{otherwise, where } x \text{ is a new variable} \quad (3) \end{cases}$$

In words: we iterate through s from beginning to end. If the characters in p and s are the same, they are also the same in q. If there are different characters in p and s, they are replaced with a variable. The same variable is used for the same combinations of characters in p and s. The resulting pattern q is in canonical form and is descriptive of $L(p) \cup \{s\}$ (lemma 2 in [9]).

Example. *We have a pattern* $p = 0xx011$ *and a string* $s = 011100$. *We construct* q*:*

i	*1*	*2*	*3*	*4*	*5*	*6*
p_i	0	x	x	0	1	1
s_i	0	1	1	1	0	0
Case	*1*	*3*	*2*	*3*	*3*	*2*
q_i	0	x_1	x_1	x_2	x_3	x_3

With this result, we may now describe the algorithm by Lange and Wiehagen.

4.3 Lange and Wiehagen's algorithm

Let $L(p) = \{s_1, s_2, \ldots\}$ the language of an unknown pattern p that is presented to a learner as a text. Lange and Wiehagen's algorithm can be used to identify p in the limit.

The learner's initial hypothesis is $h_1 = s_1$. Then, whenever a new string s_{n+1} is received, the learner uses the previous hypothesis h_n to construct a new hypothesis h_{n+1}: s_{n+1} is ignored if it is longer than h_n or adopted as the new hypothesis if it is shorter. If h_n and s_{n+1} have the same length, the learner chooses the pattern q that is descriptive of h_n and s_{n+1} as the new hypothesis.

Once the learner received a string of shortest length, the hypotheses will be patterns descriptive of the shortest length strings of $L(p)$. Thus, a correct hypothesis will be found after processing at most every minimal string from $L(p)$.

At each step i, this algorithm runs in time linear in the length of s_i. It must be noted, however, that even though this results in polynomial update time, the time needed to arrive at the correct hypothesis (i.e. the *total learning time*) is still unbounded. For example, the input text may contain the same string a million times before providing the next one. In the average case, the time needed to arrive at the correct hypothesis is exponential in the number of variables in p [13].

Example. *The strings of minimal length generated by the pattern $p = x_1 1 x_2 0 x_1$ are*

$$11101, 11001, 01100, 01000$$

Assuming the algorithm processes the strings in the above order, then

$$h_1 = 11101, h_2 = 11x_101, h_3 = x_1 1 x_2 0 x_1, h_4 = x_1 1 x_2 0 x_1$$

Clearly, the hypothesis will not change afterwards.

5 Other results

This section briefly summarizes some of the further work published on pattern languages without going into too much detail. The original definition of pattern languages by Angluin has been modified by other authors, for an overview see [12] and [10]. Possible extensions of pattern languages include *extended pattern languages*, where empty substitutions are allowed, or *regular pattern*, which contain each variable at most once. Table 2 describes the properties of these modified pattern languages. Notably, the regular patterns produce exactly the regular languages, while the equivalence problem for extended pattern languages is still being actively researched [5].

As discussed in the previous section, polynomial *update* time does not guarantee good overall *learning* time. In [4], several ways to optimize the average learning time are discussed. For example, the learning time may be optimized by dropping the condition to find only patterns of maximum possible length or by designing an algorithm that can run in parallel.

Language	Membership	Equivalence	Inclusion
Standard	NP	P	U
Regular	P	P	P
Extended	NP	Open	U
Extended Regular	P	P	P

Table 2: Properties of different types of pattern languages. U=undecidable

Aside from this, a few practical applications have been presented. These include data management systems that automatically identify the structure of the data they must store (cf. [11]) as well as methods that find patterns in amino acid sequences from positive examples (cf. [3]).

6 Conclusion

In this report, we have looked at pattern languages. These were originally introduced as a model for inductive inference of formal languages, the question being, if and how a pattern descriptive of a set of strings can be found algorithmically. We studied the results of Dana Angluin in [1], which state that finding descriptive patterns of maximum length is an NP-complete problem. We then described her algorithm for finding descriptive patterns with only one variable. Furthermore, we put the problem into the context of Gold's model of learning in the limit and discussed the algorithm by Lange and Wiehagen. Finally, we described some further results of research into pattern languages.

In the context of algorithmic learning theory, the study of pattern languages has resulted in countless theoretical insights into conditions for formal language that can be learned in the limit [10]. Historically, the papers by Angluin in 1980 affirmed that there are important classes of languages to be learned from positive data. Learning from positive examples is especially relevant, because in many situations a learner cannot access negative data on a phenomenon he is observing. This applies, for example, to the problem of finding patterns in natural structures.

The more practical discipline of machine learning builds up on theoretical foundations including those of algorithmic learning theory. We have already described some practical applications of pattern languages. Some recent research has focused on finding very efficient solutions for learning pattern languages [4]. It remains to be seen, whether these results will find applications in machine learning.

There are also some questions regarding properties of derivations from the original pattern languages that remain open until today [5]. It will be interesting to find answers for these questions through future research.

Appendix A Example automata

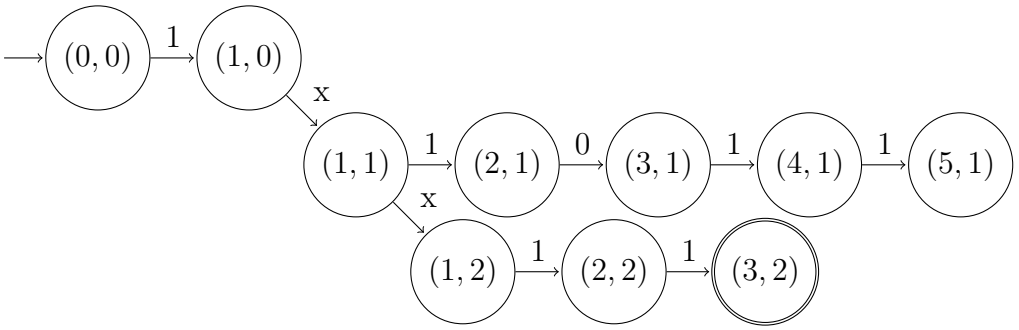

Figure 3: Automaton $B_{1101011}(3, 2, 2)$ (reduced)

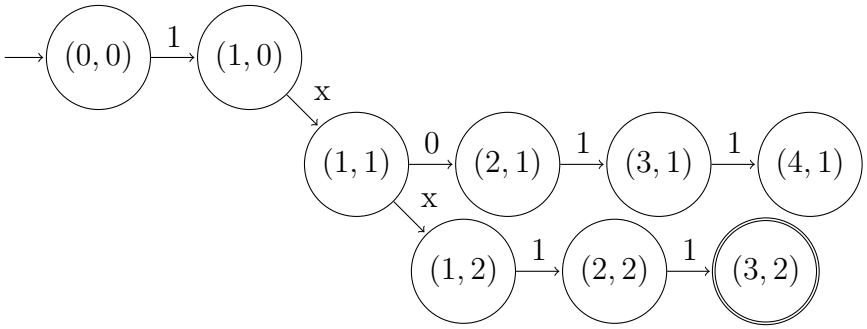

Figure 4: Automaton $B_{10011}(3, 2, 2)$ (reduced)

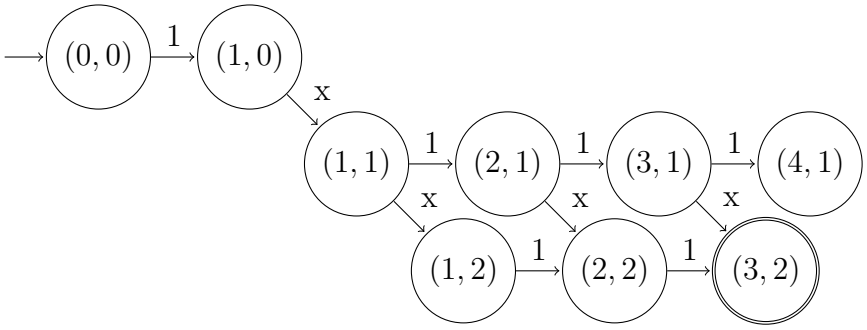

Figure 5: Automaton $B_{11111}(3, 2, 2)$ (reduced)

List of Figures

List of Tables

References

[1] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46 – 62, 1980.

[2] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117 – 135, 1980.

[3] Hiroki Arimura, Ryoichi Fujino, Takeshi Shinohara, and Setsuo Arikawa. Protein motif discovery from positive examples by minimal multiple generalization over regular patterns. *Genome Informatics*, 5:39–48, 1994.

[4] Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Agelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theor. Comput. Sci.*, 261(1):119–156, June 2001.

[5] Dominik D. Freydenberger and Daniel Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208(1):83 – 96, 2010.

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

[7] Tao Jiang, Ming Li, Bala Ravikumar, and Kenneth W. Regan. Formal grammars and languages. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook*, pages 20–20. Chapman & Hall/CRC, 2010.

[8] Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. Inclusion is undecidable for pattern languages. In Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming*, volume 700 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin Heidelberg, 1993.

[9] Steffen Lange and Rolf Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8(4):361–370, 1991.

[10] Yen Kaow Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397(1):150–165, 2008.

[11] Takeshi Shinohara. Polynomial time inference of pattern languages and its applications. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

[12] Takeshi Shinohara and Setsuo Arikawa. Pattern inference. In *Algorithmic Learning for Knowledge-Based Systems*, pages 259–291. Springer, 1995.

[13] Thomas Zeugmann. Lange and wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence*, 23(1-2):117–145, January 1998.