Grammatical Inference by Answer Set Programming[⋆]

Wojciech Wieczorek[1][0000−0003−3191−9151], Łukasz Strąk[1][0000−0002−1074−2847], Arkadiusz Nowakowski[1][0000−0001−8304−5746], and Olgierd Unold[2][0000−0003−4722−176X]

[1] Institute of Computer Science, University of Silesia in Katowice, Poland
{wojciech.wieczorek,lukasz.strak,arkadiusz.nowakowski}@us.edu.pl
[2] Department of Computer Engineering, Wrocław University of Science and Technology, Poland
olgierd.unold@pwr.edu.pl

Abstract. In this paper, the identification of context-free grammars based on the presentation of samples is investigated. The main idea of solving this problem proposed in the literature is reformulated in two different ways: in terms of general constrains and as an answer set program. In a series of experiments, we showed that our answer set programming approach is much faster than our alternative method and the original SAT encoding method. Similarly to a pioneer work, some well-known context-free grammars have been induced correctly, and we also followed its test procedure with randomly generated grammars, making it clear that using our answer set programs increases computational efficiency. The research can be regarded as another evidence that solutions based on the stable model (answer set) semantics of logic programming may be a right choice for complex problems.

Keywords: Grammatical inference · Answer set programming · Constraint satisfaction problem.

1 Introduction

In grammatical inference [9], a learning algorithm LA takes a finite sequence (usually strings) of examples as input and outputs a language description (usually grammars). There are two main types of presentations: (i) A *text* for a language L is an infinite sequence of strings $x_1, x_2, \ldots$ from L such that every string of L occurs at least once in the text; (ii) An *informant* for a language L is an infinite sequence of pairs $(x_1, d_1), (x_2, d_2), \ldots$ in $\Sigma^* \times \mathbb{B}$ such that every string of Σ^* occurs at least once in the sequence and $d_i = \text{true} \iff x_i \in L$. The inference algorithms that use type (ii) of information are said to learn from *positive and negative examples*. From the Gold's results [7], we know that the class of context-free languages (and even regular languages) cannot be identified from

[⋆] This research was supported by National Science Center (Poland), grant number 2016/21/B/ST6/02158.

presentation (i), but can be identified using presentation (ii). However, de la Higuera [8] showed that it is computationally hard.

In this work, the following informant learning environment is exploited. Suppose that the inferring process is based on the existence of an *Oracle*, which can be seen as a device that:

1. Knows the language and has to answer correctly.
2. Can answer *equivalence queries*. They are made by proposing some hypothesis to the Oracle. The hypothesis is a grammar representing the unknown language. The Oracle answers YES in the positive case. In the negative case, the Oracle has to return the shortest string in the symmetric difference between the target language and the submitted hypothesis.

Then the following procedure can be applied. Start from a small[3] sample S and $k = 1$. The parameter k denotes the number of non-terminal symbols in the target grammar. Run an answer set program (or another exact method). Every time it turns out that there is no solution that satisfies all of the constraints, increase k by 1. As long as the Oracle returns a pair (x, d) in response to an equivalent query, add (x, d) to S and run the answer set program again (or respectively another exact method). Stop after the answer is YES. Unfortunately, there is no guarantee that the procedure will terminate in a polynomial number of steps, even when the target language is regular [1]. The equivalence checking may be done by random sampling. The positive answer could be incorrect, but this probability decreases if the sampling is repeated.

A very similar procedure for the induction of context-free grammars was proposed by Imada and Nakamura [11]. However, for the exact searching of k-variable grammar, they used Boolean formulas and applied an SAT solver. We took over their main Boolean variables, treating them as predicates, and then constructed a new encoding founded on answer set programming. In an alternative approach, we used general constraints of Gurobi Optimizer[4] instead of ASP.

1.1 Related work

The most closely related work to CFG identification is by Imada and Nakamura [11]. They proposed a way to synthesize CFGs from positive and negative samples based on solving a Boolean satisfiability problem (SAT). They translated the learning problem for a CFG into a SAT, which is then solved by a SAT solver. The result of the SAT solver satisfying the SAT contains a minimal set of rules (it can be easily changed to a minimal set of variables) that derives all positive samples and no negative samples.

They used one *derivation constraint* and two main types of Boolean variables:

[3] We are aware of this imprecision. The number of words and their lengths should allow of executing a program in a reasonable amount of time. In experiments, we took two words: one example and one counter-example.

[4] https://www.gurobi.com/

derivation variables A set of derivation variables represents a relation between nonterminal symbols and substrings (in other words, derivation or parse tree) of each (positive or negative) sample w as follows: for any substring x of w and $p \in V$, the derivation variable T^p_x represents that the nonterminal p derives the string x.

rule variables A set of rule variables represents a rule set as follows: for any $p, q, r \in V$, $a \in \Sigma$, a variable R^p_{qr} (or R^p_a) determines whether the production rule $p \to q\,r$ (or $p \to a$) is a member of the set of rules or not.

The derivation constraint is a set of following Boolean expressions for any string $a_1 \cdots a_n$ $(n > 1)$ and nonterminal $p \in V$.

$$T^p_{a_1 \cdots a_n} \leftrightarrow \bigvee_{i=1}^{n-1} \bigvee_{q \in V} \bigvee_{r \in V} \left(R^p_{qr} \wedge T^q_{a_1 \cdots a_i} \wedge T^r_{a_{i+1} \cdots a_n} \right).$$

Nakamura et al. have been working on another approach for incremental learning of CFGs implemented in the Synapse system [15]. This approach is based on rule generation by analyzing the results of bottom-up parsing for positive samples and searching for rule sets. Their system can also learn similar CFGs but does it only from positive samples. Both methods synthesized similar rule sets for each language in their experiments. They reported that the computation time by the SAT-based approach is rather shorter than Synapse in most languages.

1.2 Our contribution

The purpose of the present proposal is to investigate to what extent the power of an ASP solver makes it possible to tackle the context-free inference problem for large-size instances and to compare our approach with the original one. Because of the possibility of future comparisons with other methods, the Python implementation[5] of our winning method is given via GitLab.

The main original scientific contributions are as follows:

- the formulation of the induction of a k-variable context-free grammar in terms of logical rules with answer set semantics;
- the formulation of the induction of a k-variable context-free grammar in terms of general constraints;
- the construction of an informant learning algorithm based on ASP, CSP, and SAT solvers;
- the conduct of an appropriate statistical test in order to determine the fastest CFG inference method.

This paper is organized into five sections. In section 2, we present necessary definitions and facts originating from formal languages and declarative problem-solving. Section 3 describes our inference algorithms: (a) based on solving an answer set program, and (b) based on solving a constraint satisfaction program,

[5] The Python scripting language is used only for generating appropriate AnsProlog facts.

including general constraints such as AND/OR. Section 4 shows the experimental results of our approaches in comparison with the original one. Concluding comments are made in Section 5.

2 Preliminaries

We assume the reader to be familiar with basic context-free languages theory, e.g., from [10], so that we introduce only some notations and notions used later in the paper.

2.1 Words and languages

An *alphabet* is a finite, non-empty set of symbols. We use the symbol Σ for the alphabet. A *word* is a finite sequence of symbols chosen from the alphabet. We denote the length of the word w by $|w|$. The *empty word* ε is the word with zero occurrences of symbols. Let x and y be words. Then xy denotes the *catenation* of x and y, that is, the word formed by making a copy of x and following it by a copy of y. As usual, Σ^* denotes the set of words over Σ. The word w is called a *prefix* of the word u if there is a word x such that $u = wx$. We call it a *proper* prefix if $x \neq \varepsilon$. The word w is called a *suffix* of the word u if there is a word x such that $u = xw$. It is a *proper* suffix if $x \neq \varepsilon$. A *factor* (or *subword*) is a prefix of a suffix. A set of words, all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a *language*.

2.2 Context-Free Grammars

A *context-free grammar* (CFG) is defined by a quadruple $G = (V, \Sigma, P, v_0)$, where V is an alphabet of *variables* (or sometimes *non-terminal symbols*), Σ is an alphabet of *terminal symbols* such that $V \cap \Sigma = \emptyset$, P is a finite set of *production rules* in the form $A \to \alpha$ for $A \in V$ and $\alpha \in (V \cup \Sigma)^*$, and v_0 is a special non-terminal symbol called the *start symbol*. For simplicity's sake, we write $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$ instead of $A \to \alpha_1$, $A \to \alpha_1, \ldots, A \to \alpha_k$. We call the word $x \in (V \cup \Sigma)^*$ a *sentential form*. Let u, v be two words in $(V \cup \Sigma)^*$ and $A \in V$. Then, we write $uAv \Rightarrow uxv$, if $A \to x$ is a rule in P. That is, we can substitute the word x for symbol A in a sentential form if $A \to x$ is a rule in P. We call this rewriting a *derivation*. For any two sentential forms x and y, we write $x \Rightarrow^* y$, if there exists a sequence $x = x_0, x_1, x_2, \ldots, x_n = y$ of sentential forms such that $x_i \Rightarrow x_{i+1}$ for all $i = 0, 1, \ldots, n-1$. The language $L(G)$ generated by G is the set of all words over Σ that are generated by G; that is, $L(G) = \{x \in \Sigma^* \mid v_0 \Rightarrow^* x\}$. A language is called a *context-free language* if it is generated by a context-free grammar. Assume that G is the unknown (target) CFG to be identified. An *example* (a *positive word*) of G is a word in $L(G)$, and a *counter-example* (a *negative word*) of G is a word not in $L(G)$.

A *normal form* for context-free grammars is a form, for which any grammar can be converted to the respective normal form version. Amongst all normal

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_4

forms for context-free grammars, the most useful and the most well-known one is the Chomsky normal form (CNF). A grammar is said to be in *Chomsky normal form* if each of its rules is in one of two possible forms:

(a) $X \to x$, $x \in \Sigma$, $X \in V$, or
(b) $X \to Y\,Z$, $X, Y, Z \in V$.

2.3 Answer Set Programming

We will briefly introduce the idea of answer set programming (ASP). Those who are interested in a more detailed description of the topic, alternative definitions, and the formal specification of this kind of logic programming are referred to handbooks [3], [6], and [12].

A variable or constant is a *term*. An *atom* is $a(t_1, \ldots, t_n)$, where a is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. A *literal* is either a *positive literal* p or a *negative literal* $\neg p$, where p is an atom.

A *rule* r is a clause of the form

$$a_0 \leftarrow a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m \quad m \geq 0, \tag{1}$$

where $a_0, \ldots, a_m$ are atoms. The atom a_0 is the *head* or r, while the conjunction $a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m$ is the *body* of r. By $H(r)$, we denote the head atom, and by $B(r)$ the set $\{a_1, \ldots, a_k, \neg a_{k+1}, \ldots, \neg a_m\}$ of the body literals. $B^+(r)$ ($B^-(r)$, resp.) denotes the set of atoms occurring positively (negatively, resp.) in $B(r)$. A *program* (also called ASP program) is a finite set of rules. A $\neg$-free program is called *positive*. A term, atom, literal, rule, or a program is *ground* if no variables appear in it.

Let $\mathcal{P}$ be a program. Let r be a rule in $\mathcal{P}$, a *ground instance* of r is a rule obtained from r by replacing[6] every variable X in r by constants occurring in $\mathcal{P}$. We denote the set of all the ground instances of the rules occurring in $\mathcal{P}$ by ground($\mathcal{P}$).

An *interpretation* I for $\mathcal{P}$ is a set of ground atoms. A ground positive literal A is *true* (*false*, resp.) w.r.t. I if $A \in I$ ($A \notin I$, resp.). A ground negative literal $\neg A$ is *true* (*false*, resp.) w.r.t. I if $A \notin I$ ($A \in I$, resp.).

Let r be a ground rule in ground($\mathcal{P}$). The head of r is *true* w.r.t. I if $H(r) \in I$. The body of r is *true* w.r.t. I if all body literals of r are true w.r.t. I (i.e., $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$) and is *false* w.r.t. I otherwise. The rule r is *satisfied* (or *true*) w.r.t. I if r head is true w.r.t. I or r body is false w.r.t. I.

A *model* for $\mathcal{P}$ is an interpretation M for $\mathcal{P}$ such that every rule $r \in$ ground($\mathcal{P}$) is true w.r.t. M.

Given a program $\mathcal{P}$ and an interpretation I, the *reduct* $\mathcal{P}^I$ is the set of positive rules defined as follows:

$$\mathcal{P}^I = \{H(r) \leftarrow \bigwedge B^+(r) \mid r \in \text{ground}(\mathcal{P}) \text{ and } B^-(r) \cap I = \emptyset\}. \tag{2}$$

[6] This process can be done efficiently, because many ground instances can be discarded; see Chapter 4 of [6]

I is an *answer set* of $\mathcal{P}$ if I is the $\subseteq$-smallest model for $\mathcal{P}^I$.

Over the last years, answer set programming has emerged as a declarative problem-solving paradigm. It is a programming methodology rooted in research on artificial intelligence and computational logic, and researchers use it in many areas of science and technology. For experiments we took advantages of CLINGO—one of the most efficient and widely used answer set programming system available[7] today. In addition to standard definitions, CLINGO allows to define *constraints*, i.e., rules with the empty head, for instance

$$\leftarrow a(t) \tag{3}$$

By adding this constraint to a program, we eliminate its answer sets that contain $a(t)$. Adding the 'opposite' constraint

$$\leftarrow \neg a(t) \tag{4}$$

eliminates those answers that do not contain $a(t)$. A constraint can be translated into a normal rule. To this end, the constraint

$$\leftarrow a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m \tag{5}$$

is mapped onto the rule

$$x \leftarrow a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m \wedge \neg x \tag{6}$$

where x is a new atom.

Example. Suppose we have three numbered urns and two distinguishable balls. Every ball has been put to an urn, maybe to the same. An ASP program to code this knowledge is as follows:

$$\mathrm{urn}(1) \leftarrow \tag{7}$$
$$\mathrm{urn}(2) \leftarrow \tag{8}$$
$$\mathrm{urn}(3) \leftarrow \tag{9}$$
$$\mathrm{ball}(q) \leftarrow \tag{10}$$
$$\mathrm{ball}(r) \leftarrow \tag{11}$$
$$\mathrm{contains}(U, B) \leftarrow \mathrm{urn}(U) \wedge \mathrm{ball}(B) \wedge \neg \mathrm{not\_in}(U, B) \tag{12}$$
$$\mathrm{not\_in}(U, B) \leftarrow \mathrm{urn}(U) \wedge \mathrm{urn}(V) \wedge U \neq V \wedge \mathrm{ball}(B) \wedge \mathrm{contains}(V, B) \tag{13}$$
$$\mathrm{in}(B) \leftarrow \mathrm{urn}(U) \wedge \mathrm{ball}(B) \wedge \mathrm{contains}(U, B) \tag{14}$$
$$\leftarrow \mathrm{ball}(B) \wedge \neg \mathrm{in}(B) \tag{15}$$

Please notice that as usual in logic programming, identifiers with initial uppercase letters are assigned to variables. Rules 7–11 are simple facts concerning urns and balls. Rules 12 and 13 define predicates that tell whether a ball is inside in

[7] https://potassco.org/

a particular urn. Inequality $U \neq V$ is only used during grounding to eliminate some ground instances of rule 13. It is worth mentioning that grounding systems do not make unnecessary replacements, for example, 1 for U. Rules 14 and 15 ensure that every ball is exactly in one urn.

Suppose now that we have discovered that urn 2 is empty and we want to know possible configurations. It is enough to add two facts:

$$\text{not\_in}(2, q) \leftarrow \tag{16}$$
$$\text{not\_in}(2, r) \leftarrow \tag{17}$$

and find all answer sets. A possible answer set is: ball(q), ball(r), urn(1), urn(2), in(r), not_in($2, q$), not_in($2, r$), not_in($3, q$), not_in($3, r$), contains($1, q$), in(q), urn(3), contains($1, r$), which describes the placement of both balls into the first urn.

CLINGO also allows using choice constructions, for instance:

$$\{\text{p}(U, B) \colon \text{urn}(U)\} = 2 \leftarrow \text{ball}(B) \tag{18}$$

describes all possible ways to choose which two of the atoms p($1, q$), p($2, q$), p($3, q$) and which two of the atoms p($1, r$), p($2, r$), p($3, r$) are included in the resultant model. Before and after an expression in braces, we can put integers, which express bounds on the cardinality of the stable models described by the rule. The number on the left is the lower bound (0 is default), and the number on the right is the upper bound (unbounded is default).

3 Proposed Encodings for the Induction of CFGs

Our translation converts CFG identification into an ASP program (the main approach) and CSP model (an alternative approach, constraint satisfaction problem). Suppose we are given a sample composed of examples, S_+, and counter-examples, S_-, over an alphabet Σ, and a positive integer k. We want to find a k-variable CFG $G = (V, \Sigma, P, v_0)$ such that $S_+ \subseteq L(G)$ and $S_- \cap L(G) = \emptyset$.

3.1 Using Logic Programming with Answer Set Semantics

Let F be the set of all factors (excluding the empty word) of $S_+ \cup S_-$. Let us now see how to describe the rules for the relationship between a grammar G and a sample $S_+ \cup S_-$ in terms of ASP. There are three main predicates: $y(I, J, L)$, which indicates the presence of $I \to J\,L$ in P; $w(I, Q)$, which indicates that $I \Rightarrow^* Q$, where Q represents a factor; and $z(I, A)$, which indicates the presence of $I \to A$.

1. We have the following domain specification, our facts.

$$\text{variable}(i) \leftarrow \qquad \text{for } i = 0, 1, \ldots, k-1 \tag{19}$$
$$\text{factor}(f) \leftarrow \qquad \text{for all } f \in F \tag{20}$$
$$\text{terminal}(a) \leftarrow \qquad \text{for all } a \in \Sigma \tag{21}$$
$$\text{positive}(s) \leftarrow \qquad \text{for all } s \in S_+ \tag{22}$$
$$\text{negative}(s) \leftarrow \qquad \text{for all } s \in S_- \tag{23}$$
$$\text{compose}(f, b, c) \leftarrow \qquad \text{for such } f, b, c \in F \text{ that } f = bc \tag{24}$$

2. The next rules ensure that in a grammar G a factor can or cannot be derived from a specific variable and ensure that in the grammar there is a subset of all possible productions.

$$\{\text{w}(I, F)\} \leftarrow \text{variable}(I) \wedge \text{factor}(F) \tag{25}$$
$$\{\text{z}(I, A)\} \leftarrow \text{variable}(I) \wedge \text{terminal}(A) \tag{26}$$
$$\{\text{y}(I, J, L)\} \leftarrow \text{variable}(I) \wedge \text{variable}(J) \wedge \text{variable}(L) \tag{27}$$
$$\text{w}(I, A) \leftarrow \text{variable}(I) \wedge \text{terminal}(A) \wedge \text{z}(I, A) \tag{28}$$
$$\text{z}(I, A) \leftarrow \text{variable}(I) \wedge \text{terminal}(A) \wedge \text{w}(I, A) \tag{29}$$

3. All examples should be accepted, and no counter-example can be accepted.

$$\leftarrow \text{positive}(F) \wedge \neg\text{w}(0, F) \tag{30}$$
$$\leftarrow \text{negative}(F) \wedge \text{w}(0, F) \tag{31}$$

4. For every $f \in F$ for which $|f| \geq 2$ and for every pair (b, c) $(b, c \in F)$ of such factors that $bc = f$, f can be derived from a non-terminal I if there are two non-terminals, J and L, such that b can be derived from J, c can be derived from L, and there is a production $I \to J\,L$.

$$\begin{aligned}\text{w}(I, F) \leftarrow\ & \text{variable}(I) \wedge \text{variable}(J) \wedge \text{variable}(L) \\ & \wedge \text{compose(F, B, C)} \wedge \text{y}(I, J, L) \wedge \text{w}(J, B) \wedge \text{w}(L, C)\end{aligned} \tag{32}$$

5. On the other hand, if $I \Rightarrow^* f$, then at least one such pair (J, L) should exist, that $I \to J\,L$ is in P and $J \Rightarrow^* b$ and $L \Rightarrow^* c$.

$$\begin{aligned}\leftarrow\ & \text{variable}(I) \wedge \text{factor}(F) \wedge \neg\text{terminal}(F) \wedge \text{w}(I, F) \\ & \wedge \{\text{y}(I, J, L)\colon \text{variable}(J) \wedge \text{variable}(L) \\ & \qquad \wedge \text{compose}(F, B, C) \wedge \text{w}(J, B) \wedge \text{w}(L, C)\} = 0\end{aligned} \tag{33}$$

3.2 Using General Constraints

This time, instead of predicates, w, y, and z are binary variables. We use the following constraints

$$w_{0s} = 1 \qquad \text{for all } s \in S_+ \tag{34}$$
$$w_{0s} = 0 \qquad \text{for all } s \in S_- \tag{35}$$

and

$$w_{if} \;\leftrightarrow \sum_{j,l\in K,\, bc=f} y_{ijl} \wedge w_{jb} \wedge w_{lc} \;+\; (z_{if} \text{ if } f \in \Sigma) \tag{36}$$

for each $(i,f) \in K \times F$, where $\alpha \leftrightarrow \beta$ means if $\alpha = 0$ then $\beta = 0$ and if $\alpha = 1$ then $\beta \geq 1$, and $K = \{0, 1, \ldots, k-1\}$.

4 Experimental Results

In this section, we describe some experiments comparing the performance of our approaches implemented[8] in Python, using CLINGO (ASP) and using Gurobi Optimizer, with our implementation of Imada et al. algorithm [11] using the PicoSAT solver (SAT), when positive and negative words are given. For these experiments, we use a set of 40 samples: partly based on randomly generated grammars (33 samples) and partly based on the set of fundamental CFGs appearing in grammatical inference research (the last 7 samples).

4.1 Benchmarks

For testing the learning power for general CFGs, we randomly generated 33 CFGs and prepared positive and negative samples with lengths no longer than 14 exhaustively enumerated for them. The grammars are in Chomsky normal form with 6 to 12 rules on the alphabet $\{a, b\}$. In every sample, positive words constitute not less than 20 % of the total.

The last seven samples are also with lengths no longer than 14 exhaustively enumerated, but they were generated based on the following descriptions:

(a) The set of palindromes over $\{a, b\}$.
(b) The parentheses language: the set of strings consisting of equal numbers of a's and b's such that every prefix does not have more b's than a's.
(c) The set of strings consisting of b's twice as many as a's.
(d) The set of strings of a's and b's not of the form ww.
(e) The complement of the language (b).
(f) $\{a^n b^n \mid n \geq 1\}$.
(g) The set of strings consisting of equal numbers of a's and b's.

4.2 Performance Comparison

In all experiments, we used Intel Xeon CPU E5-2650 v2, 2.6 GHz (single-core out of eight), under Ubuntu 18.04 operating system with 60 GB available RAM. Algorithm 1 shows the process for synthesizing a grammar (the set of production rules with v_0 being always the start symbol) from positive and negative words. In the algorithm, S_+ and S_- represent the set of positive and negative words

[8] https://gitlab.com/answer-set-programming/asp4cfg

Algorithm 1 Synthesize CFG G from examples and counter-examples

```
Require: $S_+$ positive words, $S_-$ negative words, $\Sigma$ a set of terminal symbols
Ensure: $G$ a context-free grammar consistent with $S_+$ and $S_-$
  $S'_+ \leftarrow \{$the shortest word from $S_+\}$
  $S'_- \leftarrow \{$the shortest word from $S_-\}$
  $k \leftarrow 1$
  loop
    $R \leftarrow$ Convert($S'_+$, $S'_-$, $\Sigma$, $k$)
    find a stable model $M$ for $R$ by the solver
    while $R$ has no stable model $M$ do
      $k \leftarrow k + 1$
      $R \leftarrow$ Convert($S'_+$, $S'_-$, $\Sigma$, $k$)
      find a stable model $M$ for $R$ by the solver
    end while
    $P \leftarrow$ Extract($M$)
    $G \leftarrow (\{v_0, v_1, \ldots, v_{k-1}\}, \Sigma, P, v_0)$
    $X \leftarrow S_+ \setminus L(G)$
    $Y \leftarrow S_- \cap L(G)$
    if $X = \emptyset$ and $Y = \emptyset$ then
      return $G$
    else
      add appropriately the shortest word from $X \cup Y$ to $S'_+$ or to $S'_-$
    end if
  end loop
```

as an input. The variables S'_+ and S'_- hold sets of samples to be covered in the next loop iteration. The algorithm picks up a word from S_+ or S_- that is not covered by the inferred grammar G, and add it to S'_+ or S'_-. The function *Convert* translates the problem into a set of ASP rules R (or Gurobi general constraints or a Boolean expression). If the ASP solver (or Gurobi Optimizer or the SAT solver) finds a stable model M, the function *Extract* returns a set of production rules by analyzing the presence of particular $\mathrm{y}(i, j, l)$ and $\mathrm{z}(i, a)$ atoms. The algorithm repeats this process—increasing k to relaxe the limit on the number of non-terminals—until G covers the all given S_+ and S_-.

The results are listed in Table 1. In order to determine whether the observed CPU time differences between ASP's runs and the remaining methods' runs did not occur by chance, we use the Wilcoxon signed-rank test [17, pp. 915–916] for ASP vs SAT and ASP vs Gurobi. The *null hypothesis* to be tested is that the median of the paired differences is negative (against the alternative that it is positive). As we can see from Table 2, p-value is high in both cases, so the null hypothesis cannot be rejected, and we may conclude that using our ASP encoding is likely to improve CPU time performance for most of this kind of benchmarks.

Table 1. Execution times of exact solving CFG identification in seconds

Language	$\|V\|$	ASP	SAT	Gurobi
1	3	51.70	48.65	56.42
2	6	646.39	21049.22	$>$ 21050
3	4	74.31	189.85	143.76
4	5	75.90	347.84	$>$ 2000
5	4	27.91	64.82	18.36
6	5	75.96	335.98	10.33
7	4	68.35	61.87	$>$ 2000
8	4	57.14	118.25	28.85
9	3	45.17	94.86	73.03
10	5	211.33	568.12	568.06
11	5	62.48	166.65	$>$ 2000
12	3	21.50	58.12	33.28
13	6	112.69	705.80	$>$ 2000
14	6	943.02	4807.32	$>$ 4808
15	7	19358.09	252290.70	$>$ 252291
16	4	49.01	111.22	103.05
17	7	2921.44	8035.44	$>$ 8036
18	5	361.52	1369.22	$>$ 2000
19	5	63.47	238.71	186.10
20	2	12.96	5.64	3.88
21	5	96.68	512.83	671.62
22	2	11.38	12.02	10.54
23	3	11.84	43.03	9.92
24	4	109.98	159.73	176.49
25	3	22.65	22.40	29.65
26	5	38.74	271.30	420.11
27	5	94.76	295.81	$>$ 2000
28	5	216.61	625.07	$>$ 2000
29	5	271.88	324.43	$>$ 2000
30	6	228.98	412.16	$>$ 2000
31	2	10.97	15.29	19.84
32	5	62.17	293.98	105.74
33	3	10.42	18.30	13.15
34	5	31.13	49.28	32.83
35	3	12.84	20.97	12.86
36	4	118.17	76.98	73.74
37	6	173.66	191.42	$>$ 2000
38	4	29.33	54.63	36.71
39	4	4.12	21.00	9.02
40	3	66.71	50.65	40.40

Table 2. Obtained p-values from the Wilcoxon signed-rank test

ASP vs SAT	ASP vs Gurobi
0.999999647	0.999987068

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_4

4.3 ASP-based CFG induction on bioinformatics datasets

Our induction method can also be applied to other data, that are not taken from context-free infinite languages. We tried its classification quality on two bioinformatics datasets: WALTZ-DB database [4], composed by 116 hexapeptides known to induce amyloidosis (S_+) and by 161 hexapeptides that do not induce amyloidosis (S_-) and Maurer-Stroh et al. database from the same domain [14], where the ratio of S_+/S_- is 240/836.

We chose a few standard machine learning methods for comparison: BNB (Naive Bayes classifier for multivariate Bernoulli models [13, pp. 234–265]), DTC (Decision Trees Classifier, CART method [5]), MLP (Multi-layer Perceptron [16]), and SVM (Support Vector Machine classifier with the linear kernel [18]). In all methods except ASP and BNB, an unsupervised data-driven distributed representation, called ProtVec [2], was applied in order to convert words (protein representations) to numerical vectors. For using BNB, we represented words as binary-valued feature vectors that indicated the presence or absence of every pair of protein letters. In case of ASP, the training set was partitioned randomly into n parts, and the following process was being performed m times. Choosing one part for synthesizing a CFG and use rest $n-1$ parts for validating it. The best of all m grammars—in terms of higher F-measure—was then confronted with the test set. For WALTZ-DB n and m have been set to 20, for Maurer-Stroh n has been set to 10 and m to 30. These values were selected experimentally based on the size of databases and the running time of the ASP solver.

To estimate the ASP's and compared approaches' ability to classify unseen hexapeptides repeated 10-fold cross-validation (cv) strategy was used. It means splitting the data randomly into 10 mutually exclusive folds, building a model on all but one fold, and evaluating the model on the skipped fold. The procedure was repeated 10 times and the overall assessment of the model was based on the mean of those 10 individual evaluations. Table 3 summarizes the performances of the compared methods on WALTZ-DB and Maurer-Stroh databases. It is noticable that the ASP approach achieved best F-score for smaller dataset

Table 3. Performance of compared methods on WALTZ-DB and Maurer-Stroh databases in terms of Precision (P), Recall (R), and F-score (F1)

Method	WALTZ-DB			Maurer-Stroh		
	P	R	F1	P	R	F1
ASP	0.38 ± 0.09	0.58 ± 0.12	0.45 ± 0.07	0.58 ± 0.12	0.66 ± 0.18	0.61 ± 0.12
BNB	0.51 ± 0.09	0.69 ± 0.14	0.59 ± 0.10	0.61 ± 0.11	0.60 ± 0.13	0.60 ± 0.11
DTC	0.43 ± 0.11	0.59 ± 0.26	0.46 ± 0.11	0.36 ± 0.20	0.74 ± 0.39	0.48 ± 0.26
MLP	0.49 ± 0.20	0.57 ± 0.27	0.46 ± 0.10	0.43 ± 0.09	0.90 ± 0.07	0.58 ± 0.10
SVM	0.37 ± 0.06	0.69 ± 0.07	0.48 ± 0.06	0.24 ± 0.21	0.51 ± 0.44	0.32 ± 0.28

(Maurer-Stroh) and an average F-score for the bigger one (WALTZ-DB), hence it can be used with a high reliability to recognize amyloid proteins. BNB is outstanding for the WALTZ-DB and almost as good as ASP for Maurer-Stroh database.

5 Conclusion

In this paper, we proposed an approach for learning context-free grammars from positive and negative samples by using logic programming. We encode the set of samples, together with limits on the number of non-terminals to be synthesized as an answer set program. A stable model (an answer set) for the program contains a set of grammar rules that derives all positive samples and no negative samples. A feature of this approach is that we can synthesize a compact set of rules in Chomsky normal form. The other feature is that our learning method reflects future improvements on ASP solvers. We present experimental results on learning CFGs for fundamental context-free languages, including a set of strings composed of the equal numbers of a's and b's and the set of strings over $\{a, b\}$ not of the form ww. Another series of experiments on random languages shows that our encoding can speed up computations in comparison with SAT and CSP encodings.

References

1. Angluin, D.: Negative results for equivalence queries. Machine Learning **5**(2), 121–150 (1990)
2. Asgari, E., Mofrad, M.R.K.: Continuous distributed representation of biological sequences for deep proteomics and genomics. PLOS ONE **10**(11), 1–15 (11 2015). https://doi.org/10.1371/journal.pone.0141287
3. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2003)
4. Beerten, J., Van Durme, J., Gallardo, R., Capriotti, E., Serpell, L., Rousseau, F., Schymkowitz, J.: Waltz-db: a benchmark database of amyloidogenic hexapeptides. Bioinformatics **31**(10), 1698–1700 (2015)
5. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA (1984)
6. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool Publishers (2012)
7. Gold, E.M.: Language identification in the limit. Information and Control **10**, 447–474 (1967)
8. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning **27**(2), 125–138 (1997)
9. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)
10. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, second edn. (2001)
11. Imada, K., Nakamura, K.: Learning context free grammars by using SAT solvers. In: Proceedings of the 2009 International Conference on Machine Learning and Applications. pp. 267–272. IEEE Computer Society (2009)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_4

12. Lifschitz, V.: Answer Set Programming. Springer (2019)
13. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK (2008)
14. Maurer-Stroh, S., Debulpaep, M., Kuemmerer, N., de la Paz, M.L., Martins, I.C., Reumers, J., Morris, K.L., Copland, A., Serpell, L., Serrano, L., Schymkowitz, J.W.H., Rousseau, F.: Exploring the sequence determinants of amyloid structure using position-specific scoring matrices. Nature Methods **7**, 237–242 (2010)
15. Nakamura, K., Matsumoto, M.: Incremental learning of context free grammars based on bottom-up parsing and search. Pattern Recognintion **38**(9), 1384–1392 (2005). https://doi.org/10.1016/j.patcog.2005.01.004
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
17. Salkind, N.J.: Encyclopedia of Research Design. SAGE Publications, Inc (2010)
18. Wu, T.F., Lin, C.J., Weng, R.C.: Probability estimates for multi-class classification by pairwise coupling. Journal of Machine Learning Research **5**, 975–1005 (2004)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_4