Three Learnable Models for the Description of Language

Alexander Clark

Presentation by Peter Černo

About

- **Introduction**
  - **Representation classes** should be defined in such a way that they are **learnable**.
- **1. Canonical deterministic finite automata**
  - The **states** of the automaton correspond to right **congruence classes** of the language.
- **2. Context free grammars**
  - The **non-terminals** of the grammar correspond to the **syntactic congruence classes**.
- **3. Residuated lattice structure**
  - From the **Galois connection** between strings and contexts, called the **syntactic concept lattice**.

Introduction

- **Formal Language Theory** (FLT)
  - Has its **roots** in the **modeling of learning** and of **language**.
  - Originates from **linguistics**. Yet it has moved **far from its origins**.
  - Now it is an **autonomous part of computer science**, and only few papers at the major conferences in FLT are directly concerned with linguistics.

Introduction

- **Learnability**
  - The **original intention** was for **phrase-structure grammars** (**PSG**s) to be **learnable**.
  - The **PSG**s were meant to represent, at a suitable level of abstraction, the **linguistics knowledge of language**.
  - **Chomsky says**:
    - The concept of "phrase structure grammar" was explicitly designed to express the richest system that could reasonable be expected to result from the application of Harris-type procedures to a corpus…
  - "**Harris-type procedures**" refer to the **methods of distributional learning** developed by Zellig Harris.

Introduction

- **Learnability**
  - **PSG**s in general, and **CFG**s in particular, **were intended** to be **learnable** by distributional methods.
  - **But they were not**.
  - The problem **is not** with **distributional methods**.
  - The problem **is** with these **formalisms**.
- The **natural question** therefore is:
  - Are there **other formalisms**, different from Chomsky hierarchy, that are **learnable**?

Introduction

- **What we mean by learning?**
  - We construct our **representation** for the language **from information about language**.
  - We need to:
    - **Define** **representations**.
    - **Define** **algorithms** for constructing these representations.
    - **Prove**, under a suitable regime, that these algorithms will **converge** to the right answer.
  - We assume a **very good source of information**:
    - We have **positive data** and **membership queries**.
  - We consider only **algorithms** that are **efficient**.

Introduction

- **Why is learning important?**
  - **1st domain**: We have information **about the language**, but **not about the representation**.
    - **Not only linguistics**.
    - **Engineering domains**:
      - We have some **data** that we want to **model**.
      - **Computational biology** – strings of bases, amino acids.
      - **Robotics** – sequences of actions, sequences of events.
    - **Learnability is essential**!
  - **2nd domain**: We have direct information **about the representation**.
    - **Programming languages, mark up languages**:
      - We know the structure of the language.

How

- **Slogan**: "**Put learnability first!**"
- **Basic strategy**:
  - **Representations** are **objective** or "**empiricist**".
  - **Basic elements** (states, non-terminals) must have a clear **definition** in terms of **sets of strings**.
  - **Rather** than defining a function from the presentation to the language, **we should go backwards**.
  - We should define the map **from the language to the representation**.

Example

- **From** representation *G* **to** the language *L(G)*.
  - In a ***CFG***, we define a **derivation relation ⇒***.
  - For each non-terminal *N* we define:

    $L(N) = \{ w \mid N \Rightarrow^* w \}$.
  - **Result**: map from the set of ***CFG****s* to the set of ***CFL****s*.
- There is however an obstacle to going in the **reverse direction**.
  - Consider *CFL L*, and a grammar *G: L(G) = L*.
  - If *N* is a non-terminal in *G*, what constraints are there on *L(N)*?
  - We can say literally **nothing** about this set, other than that it is a context free language.

Canonical DFA

- We start by considering **regular languages**.
- We end up with the class of representations equivalent to a **subclass of *DFA***.
- **Notation**:
  - Σ – finite nonempty **alphabet**.
  - Σ^* – free monoid with λ the **empty string**.
  - A **language** L is a subset of Σ^*.
  - The **residual language** of a given string u is: $u^{-1}L = \{w \mid uw \in L\}$.
  - The following relation: $u \sim_L v$ **iff** $u^{-1}L = v^{-1}L$ is an **equivalence relation** and **right congruence**: **if** $u \sim_L v$ **and** $w \in \Sigma^*$ **then** $uw \sim_L vw$.

Canonical DFA

- **Notation**:
  - We will write $[u]^R$ for the **congruence class** of the string u under this **right congruence** $\sim_L$.
  - It is better to consider **pair** $\langle P, S \rangle$, where:
    - P is a **congruence class**,
    - S is the **residual language** of all strings in P.
  - We will have **elements** of the form:

    $\langle [u]^R, u^{-1}L \rangle$.
  - One important **element** is:

    $\langle [\lambda]^R, L \rangle$.

Canonical DFA

- **Representation** based on **congruence classes**:
  - **States** – primitive **elements** of our representation.
  - **The state** $q_0 = <[\lambda]^R, L>$.
- **Observations**:
  - **If** $u \in L$ **then every element of** $[u]^R$ is also in L.
  - **Final state** is $<P, S>$ such that $\lambda \in S$.
  - If we can tell for each string which congruence class it is in, then we will have **predicted the language**.
- **Idea**:
  - We will try to compute **for each string** w which **congruence class** it is in.

Canonical DFA

- We have defined the **primitive elements**.
- Now we have to define a **derivation**.
- **Observation**:
  - If we have a string that **we know** is in the congruence class $[u]^R$ and we **append** the string v we know that it will be in the class $[uv]^R$.
  - We can **restrict** ourselves to the case where $|v|=1$.
  - We now have something that looks very like an **automaton**.
  - We have defined a **function from** L **to** $\mathcal{R}(L)$.

Canonical DFA

- **The representation $\mathcal{R}(L)$ consists of**:
  - Q – possibly infinite set of all these **states**,
  - q_0 – the **initial state**,
  - δ – the **transition function** defined by:

    $\delta([u]^{\mathcal{R}}, a) = [ua]^{\mathcal{R}}$,
  - F – the set of **final states** $\{[u]^{\mathcal{R}} \mid u \in L\}$.
- We can define the function **from** the representation $\mathcal{R}(L)$ **to** the language $L(\mathcal{R}(L))$:

  $L(\mathcal{R}(L)) = \{ w \mid \delta(q_0, w) \in F \}$.
- For **any** language L: $L(\mathcal{R}(L)) = L$.
- **Myhill-Nerode Theorem**:
  - $\mathcal{R}(L)$ is **finite iff** L is **regular**.

Canonical DFA

- It is possible to infer theses representations for **regular languages**, using a number of different techniques depending on the details of the **source of information about the language**.
- **For instance**:
  - If we have **membership** and **equivalence queries**, we can use **Dana Angluin's L* algorithm**.
  - **Membership query** is that a teacher has to decide whether to **accept** or **reject** a given word.
  - **Equivalence query** is that a teacher gets a **conjecture** (**DFA**) and he has to decide whether this DFA is a desired DFA or not. If it is not then he also has to provide a **counterexample**.

CFGs with Congruence Classes

- We move to representations capable of representing **context-free languages**.
- We use the idea of **distributional learning**.
- These techniques were originally described by **structuralist linguists**.
- **Notation**:
  - **Context** (l, r), where $l, r \in \Sigma^*$.
  - **Operation** $\odot$: $(l, r) \odot u = lur$.
  - u **occurs in a context** (l, r) in $L \subseteq \Sigma^*$ if $lur \in L$.
  - (L, R), (L, r) refer to the obvious sets of contexts: $L \times R$, $L \times \{r\}$, and so on.

CFGs with Congruence Classes

- **Notation**:
  - **Distribution** of a string w in a language L:

    $C_L(w) = \{ (l, r) \mid lwr \in L \}$.
  - We **extend the operation** $\odot$ to contexts:

    $(l, r) \odot (x, y) = (lx, yr)$.
  - $\odot$ is obviously an **associative operation**.
- **Definition**:
  - Strings u and v are **syntactically congruent** iff they have the same **distribution**:

    $u \equiv_L v$ iff $C_L(u) = C_L(v)$.
  - We write $[u]$ for the **congruence class of** u.

CFGs with Congruence Classes

- **Classical result**:
  - The number of congruence classes is **finite** if and only if the language is **regular**.
- Our **primitive elements** will correspond to these **congruence classes.**
- **Problem**:
  - We will be restricted to **regular languages**, since we are interested in **finite representations**.
- This turns out **not to be the case**.

CFGs with Congruence Classes

- **Empty context** (λ, λ) has a special significance:
  - $(\lambda, \lambda) \in C_L(u)$ means that $u \in L$.
- If we can **predict** the **congruence class** of a string, we will know the **language**.
- We can now proceed to **derivation rules**.
- The relation $\equiv_L$ is a **congruence**:
  - If $u \equiv_L v$ then $xuy \equiv_L xvy$.
- If we take any $u' \in [u]$ and $v' \in [v]$ then $u'v' \in [uv]$.
  - $u'v \equiv_L uv$ and $u'v' \equiv_L u'v$ implies $u'v' \equiv_L uv$.
- We get **context-free productions**: $[uv] \rightarrow [u][v]$.
- And **productions**: $[a] \rightarrow a$, $[\lambda] \rightarrow \lambda$.

CFGs with Congruence Classes

- **The representation $\Phi(L)$ consists of**:
  - Set of **congruence classes** $[u]$ (possibly infinite),
  - Set of **productions**:
    - $\{ [uv] \rightarrow [u][v] \mid u, v \in \Sigma^* \}$,
    - $\{ [a] \rightarrow a \mid a \in \Sigma \}$,
    - $[\lambda] \rightarrow \lambda$.
  - Set of **initial symbols** I:
    - $I = \{ [u] \mid u \in L \}$.
  - We define **derivation** as in a *CFG*.
    - **Apparently**: $[w] \Rightarrow^* v$ iff $v \in [w]$.
  - We define $L(\Phi(L)) = \{ w \mid \exists N \in I: N \Rightarrow^* w \}$.
    - **Apparently**: $L(\Phi(L)) = L$.

CFGs with Congruence Classes

- We have used the following **schemas**:
  - *[uv] → [u][v]*, *[a] → a*, *[λ] → λ*.
  - This looks something like a **context-free grammar** in **Chomsky normal form**.
- We can have **different schemas**:
  - **Finite** grammars: *[w] → w*.
  - **Linear** grammars: *[lwr] → l[w]r*.
  - **Regular** grammars: *[aw] → a[w]*.
- **Invariant**:
  - These schemas will only derive strings of the same **congruence class**.

CFGs with Congruence Classes

- There are two **differences**:
  - We may have **more than one start symbol**.
  - If the language is **not regular** then the number of congruence classes will be **infinite**.

    Consider $L_{ab} = \{ a^n b^n \mid n \geq 0 \}$.

    If $i \neq j$ then a^i is **not congruent** to a^j.
- **Let us suppose** that:
  - We **maintain the structure** of the representation.
  - But only take a **finite set of congruence classes** V consisting of the classes corresponding to a **finite set of strings** K: $V = \{ [u] \mid u \in K \}$.
- This gives us a **finite representation** $\Phi(L, K)$.

CFGs with Congruence Classes

- If we have only **finite subset** of productions, then: $[w] \Rightarrow^* v$ **only implies** $v \in [w]$.
  - **Therefore**: $L(\Phi(L, K)) \subseteq L$.
- **<u>The class we can represent is</u>**:

  $\mathfrak{L}_{CCFG} = \{ L \mid \exists \text{ finite } K \subset \Sigma^*: L(\Phi(L, K)) = L \}$.
  - This class includes all **regular languages**.
  - It also includes **some** non-regular **context-free languages**. For L_{ab}: $K = \{ \lambda, a, b, ab, aab, abb \}$.
  - The language $L = \{ a^n b^m \mid n < m \}$ **is not** in $\mathfrak{L}_{CCFG}$, as L is the union of infinite number of congruence classes.
  - By restricting non-terminals to correspond to the congruence classes, we **lose** a bit of representational **power**, but we **gain efficient learnability**.

Back to Regular Languages

- Let A be the **minimal** DFA for a language L.
- Let Q be the **set of states** of A and $n = |Q|$.
- A string w defines a **function** f_w from Q to Q:
  $f_w(q) = \delta(q, w)$.
- There are n^n **possible such functions**.
- **If** $f_u = f_v$ **then** $u \equiv_L v$, thus there are at most n^n **possible congruence classes**.
- Holzer and Konig: we **can approach** this bound.
- Using **one non-terminal per congruence** class could be an **expensive mistake**.
- There is often some **non-trivial structure**.

Back to Regular Languages

- **Congruence classes** correspond to **functions**.
- It seems reasonable to represent them using some **basis functions**.
- If we represent each **congruence class** as $n \times n$ **Boolean matrix** T: T_{ij} **is** 1 **iff** $f_u : q_i \longmapsto q_j$,
- Then the **basis functions** are the n^2 **matrices** that have just a **single** 1.
- **Rather than having** a very large number of **very specific rules** that show how individual congruence classes combine, **we can have** a very much smaller set of **more general rules**.
- **Elements** = **sets of congruence classes**.

Distributional Lattice Grammars

- A **congruence class** $[u]$ defines the **distribution** $C_L(u)$ and vice versa.
- It is natural to consider therefore as our primitive elements **ordered pairs** $\langle S, C \rangle$ where:
  - S is a subset of Σ^*.
  - C is a subset of $\Sigma^* \times \Sigma^*$.
- Given a language L we will consider only those pairs that satisfy **two conditions**:
  - $C \odot S$ is a **subset** of L.
  - Both of these sets are **maximal**.
- If a pair $\langle S, C \rangle$ satisfies these conditions, then we call it a **syntactic concept of the language**.

Galois Connection

- Another way is to consider **Galois connection** between the sets of **strings** and **contexts**.
  - For a given language L we can define **maps** from sets of **strings** to sets of **contexts** and vice versa.
  - Given a set of strings S we can define a set of contexts S' as $S' = \{(l, r) : \forall w \in S \ lwr \in L\}$.
  - Dually we can define for a set of contexts C the set of strings C' as $C' = \{w : \forall (l, r) \in C \ lwr \in L\}$.
- A **concept** is then an ordered pair $<S, C>$ such that: $S' = C$ **and** $C' = S$.
- The **most important point** here is that these are **closure operations**: $S''' = S'$ and $C''' = C'$.

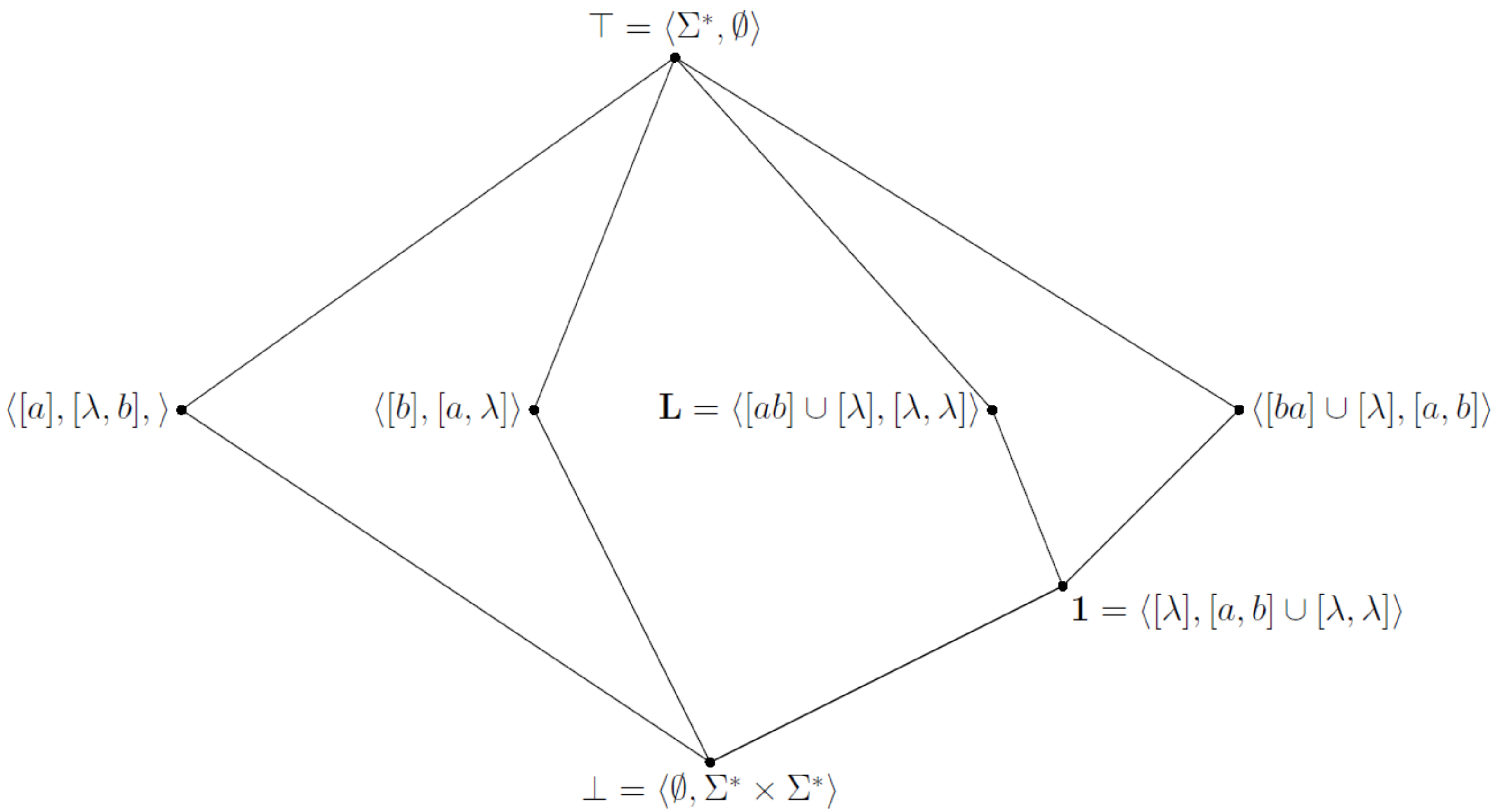
Basic Properties

- We write $\mathcal{C}(S)$ for $\langle S'', S' \rangle$ and $\mathcal{C}(C)$ for $\langle C', C'' \rangle$.
- There is an **inverse relation** between the size of the set of **strings** S and the set of **contexts** C:
  - **The larger** that S is **the smaller** that C is.
  - **In the limit** there is a concept $\mathcal{C}(\Sigma^*)$; normally this will have $C = \emptyset$.
  - **Conversely** we will always have $\mathcal{C}(\Sigma^* \times \Sigma^*)$.
- One **important concept** is $\mathcal{C}(L) = \mathcal{C}(\{(\lambda, \lambda)\})$.
- The **set of concepts** is a **partially ordered set**.
- We can **define**: $\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle$ **iff** $S_1 \subseteq S_2$.
- **Apparently**: $S_1 \subseteq S_2$ **iff** $C_1 \supseteq C_2$.

Syntactic Concept Lattice

- This partial order is a **complete lattice** $\mathcal{B}(L)$, called **syntactic concept lattice**.
  - **Topmost element** is: $\top = \mathcal{C}(\Sigma^*)$.
  - **Bottommost element** is: $\bot = \mathcal{C}(\Sigma^* \times \Sigma^*)$.
  - **Meet operation**: $\langle S_1, C_1 \rangle \wedge \langle S_2, C_2 \rangle$ can be defined as: $\langle S_1 \cap S_2 , (S_1 \cap S_2)' \rangle$.
  - **Join operation**: $\langle S_1, C_1 \rangle \vee \langle S_2, C_2 \rangle$ can be defined as: $\langle (C_1 \cap C_2)', C_1 \cap C_2 \rangle$.
- The following figure shows the syntactic concept lattice for the regular language $L = \{ (ab)^* \}$.
- L is infinite, but the lattice $\mathcal{B}(L)$ is finite.

Figure - Syntactic Concept Lattice



Monoid Structure

- Crucially, this lattice structure also has a **monoid structure**.
  - We can define a **binary operation**:

    $<S_1, C_1> \circ <S_2, C_2> = \mathcal{C}(S_1 S_2)$.
  - Operation $\circ$ is **associative** and has a **unit** $\mathcal{C}(\lambda)$.
  - Moreover, it is **monotonic**:

    If $X \leq Y$ then $X \circ Z \leq Y \circ Z$.
- We can also define **residual operations**, so this syntactic concept lattice becomes a so-called **residuated lattice**.

Representation

- Having defined and examined the syntactic concept lattice, we can now define a **representation** based on this.
- Again, if the language is not regular, the lattice will be infinite.
- We will start by considering how we might define a representation **given the whole lattice**.
- We want to be able to compute for every string w, the concept of w, $\mathcal{C}(w)$.
- If $\mathcal{C}(w) \leq \mathcal{C}(L)$ then we know that $w \in L$.
- If we know the whole lattice, then the computation of $\mathcal{C}(w)$ is quite easy.

Representation

- However, if we have a non-regular language, then we will need to restrict the lattice.
- We can do this by taking a **finite set of contexts** $F \subseteq \Sigma^* \times \Sigma^*$, which will include (λ, λ).
- This gives us a finite lattice $\mathcal{B}(L, F)$, which will have at most $2^{|F|}$ elements.
- **Lattice** $\mathcal{B}(L, F)$ is the lattice of concepts $\langle S, C \rangle$ where $C \subseteq F$, and where $C = S' \cap F$, and $S = C'$.
- We can define **concatenation** $\circ$ as before:

$$\langle S_1, C_1 \rangle \circ \langle S_2, C_2 \rangle = \langle ((S_1 S_2)' \cap F)', (S_1 S_2)' \cap F \rangle$$

- This is however **no longer** a residuated lattice.

Issues with Finite Lattice

- The operation $\circ$ is **no longer associative**.
- There may **not be** an **identity element**.
- **Nor** are the residuation operations well defined.
- However, we should still be able to **approximate the computation**.
- For some languages, and for some set of features the approximation **will be accurate**.
- It is no longer the case, that: $\mathcal{C}(u) \circ \mathcal{C}(v) = \mathcal{C}(uv)$.
- However, we can prove that: $\mathcal{C}(u) \circ \mathcal{C}(v) \geq \mathcal{C}(uv)$.
- This means that given some string w, we can compute an **upper bound** on $\mathcal{C}(w)$ quite easily.

Upper Bound

- We will call this **upper bound** $\phi(w)$.
- It may not give us exactly the right answer but it will sill be useful.
- If the upper bound $\phi(w)$ is **below** $\mathcal{C}(L)$ then we know that the string w will be in the language.
- In fact, we can compute **many different upper bounds**: since the operation $\circ$ is not associative.
- By using effective **dynamic programming** algorithm we can compute the **lowest possible upper bound** $\phi(w)$ in $O(|w|^3)$.

Lowest Possible Upper Bound

- Given a language L and set of contexts F we define $\phi: \Sigma^* \rightarrow \mathcal{B}(L, F)$ recursively by:
  - $\phi(\lambda) = \mathcal{C}(\lambda)$,
  - $\phi(a) = \mathcal{C}(a)$ for all $a \in \Sigma$,
  - for all w with $|w| > 1$,

$$\phi(w) = \bigwedge \{ \phi(u) \circ \phi(v) \mid u, v \in \Sigma^+, uv = w \}$$

- We can define the language **generated by this representation** to be:

$$L(\mathcal{B}(L, F)) = \{ w \mid \phi(w) \leq \mathcal{C}((\lambda, \lambda)) \}$$

- For any language L and any set of contexts F:

$$L(\mathcal{B}(L, F)) \subseteq L$$

Distributional Lattice Grammars

- As we **increase the set of contexts**, the **language** defined **increases monotonically**.
- In the **infinite limit** when $F = \Sigma^* \times \Sigma^*$ we have:

  $$L(\mathfrak{B}(L, \Sigma^* \times \Sigma^*)) = L$$

- We can define a natural class of languages as those which are represented by finite lattices.
- We will call this class the **Distributional Lattice Grammars (DLGs)**.
- The corresponding class of languages is:

$$\mathfrak{L}_{DLG} = \{ L \mid \exists \text{ finite set } F \subseteq \Sigma^* \times \Sigma^* : L(\mathfrak{B}(L, F)) = L \}$$

Distributional Lattice Grammars

- $\mathfrak{L}_{DLG}$ **properly includes** $\mathfrak{L}_{CCFG}$.
- $\mathfrak{L}_{DLG}$ includes some **non-context free** languages.
- $\mathfrak{L}_{DLG}$ also includes much larger set of **context free languages** than $\mathfrak{L}_{CCFG}$ including some **non-deterministic** and **inherently ambiguous** languages.
- A problem is that lattices can be **exponentially large**. We can however represent them **lazily** using a limited set of examples.
- An important future direction of research is to exploit the **algebraic structure of the lattice** to find **more compact representations**.

References

- **Clark, A., Three learnable models for the description of language**

  in Language and Automata Theory and Applications, edited by A.-H. Dediu, H. Fernau, and C. Martn-Vide, vol. 6031 of Lecture Notes in Computer Science, pp. 16 - 31, Springer Berlin / Heidelberg, 2010.