On the Complexity of Language Membership for Probabilistic Words

Antoine Amarilli[1] Mikaël Monet[1] **Paul Raphaël**[2] Sylvain Salvati[1]

[1]D-DAL Team, Inria Lille
[2]ENS Ulm & University of Cyprus

STACS 2026

Introduction and Example

We consider **probabilistic words**:

- Probabilistic word p of length n over $\Sigma = \{a, b\}$: sequence $p = p_1 \cdots p_n$ of probability distributions over letters of Σ.
  $\rightarrow$ For instance, for all $1 \leq i \leq n$, set $p_i(a) := 0.6$ and $p_i(b) := 0.4$
- Defines a probability distribution over Σ^n assuming independence.

Introduction and Example

We consider **probabilistic words**:

- Probabilistic word p of length n over $\Sigma = \{a, b\}$: sequence $p = p_1 \cdots p_n$ of probability distributions over letters of Σ.
  $\rightarrow$ For instance, for all $1 \leq i \leq n$, set $p_i(a) := 0.6$ and $p_i(b) := 0.4$
- Defines a probability distribution over Σ^n assuming independence.

Fix a language $L \subseteq \Sigma^*$, e.g., the language L_0 of words with an even number of a's.

Introduction and Example

We consider **probabilistic words**:

- Probabilistic word p of length n over $\Sigma = \{a, b\}$: sequence $p = p_1 \cdots p_n$ of probability distributions over letters of Σ.
  $\rightarrow$ For instance, for all $1 \leq i \leq n$, set $p_i(a) := 0.6$ and $p_i(b) := 0.4$
- Defines a probability distribution over Σ^n assuming independence.

Fix a language $L \subseteq \Sigma^*$, e.g., the language L_0 of words with an even number of a's.

Question

Given the probabilistic word p, compute the probability that a random word generated by p belongs to L? i.e., compute:

$$p(L) := \sum_{w \in L \cap \Sigma^n} p(w)$$

Introduction and Example

We consider **probabilistic words**:

- Probabilistic word p of length n over $\Sigma = \{a, b\}$: sequence $p = p_1 \cdots p_n$ of probability distributions over letters of Σ.
  $\rightarrow$ For instance, for all $1 \leq i \leq n$, set $p_i(a) := 0.6$ and $p_i(b) := 0.4$
- Defines a probability distribution over Σ^n assuming independence.

Fix a language $L \subseteq \Sigma^*$, e.g., the language L_0 of words with an even number of a's.

Question

Given the probabilistic word p, compute the probability that a random word generated by p belongs to L? i.e., compute:

$$p(L) := \sum_{w \in L \cap \Sigma^n} p(w)$$

Ex: for p above and L_0, we have $p(L_0) = \sum_{0 \leq j \leq \lfloor n/2 \rfloor} \binom{n}{2j} 0.6^{2j} 0.4^{n-2j}$

Probabilistic Membership Problem

For any fixed language L, define the **probabilistic membership problem** $\#\mathrm{pM}(L)$:

Problem $\#\mathbf{pM}(L)$

- **Input:** a probabilistic word p of length n
- **Output:** the probability value $p(L) = \sum_{w \in L \cap \Sigma^n} p(w)$.

Note: we only study complexity as a function of p, with L fixed (= data complexity)

Probabilistic Membership Problem

For any fixed language L, define the **probabilistic membership problem** $\#\mathrm{pM}(L)$:

Problem $\#\mathrm{pM}(L)$

- **Input:** a probabilistic word p of length n
- **Output:** the probability value $p(L) = \sum_{w \in L \cap \Sigma^n} p(w)$.

Note: we only study complexity as a function of p, with L fixed (= data complexity)

- **Naïve algorithm** (exponential):
  - Consider every possible word $w = a_1 \cdots a_n$ of Σ^n
  - Compute its probability $p(w) := \prod_{1 \leq i \leq n} p_i(a_i)$
  - Test whether $w \in L$
  - Sum the $p(w)$ for all such words

→ **Research question**: For which languages L is $\#\mathrm{pM}(L)$ in PTIME?

Roadmap

Tractability for Regular Languages and Unambiguous CFLs

Intractability for Some Inherently Ambiguous Context-Free Languages

Other Methods for Tractability

Tractability for Regular Languages and Unambiguous CFLs

Tractability for Regular Languages

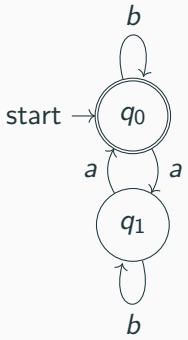
Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:

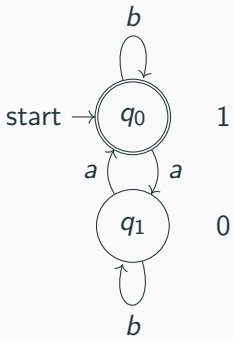


$$p_1(a) = 0.3 \quad p_2(a) = 1 \quad p_3(a) = 0.5 \quad p_4(a) = 0.3$$
$$p_1(b) = 0.7 \quad p_2(b) = 0 \quad p_3(b) = 0.5 \quad p_4(b) = 0.7$$

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:

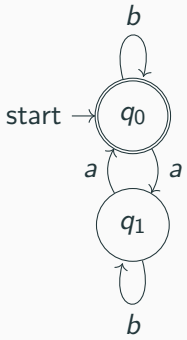


$p_1(a) = 0.3$ $\quad p_2(a) = 1$ $\quad p_3(a) = 0.5$ $\quad p_4(a) = 0.3$
$p_1(b) = 0.7$ $\quad p_2(b) = 0$ $\quad p_3(b) = 0.5$ $\quad p_4(b) = 0.7$

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:



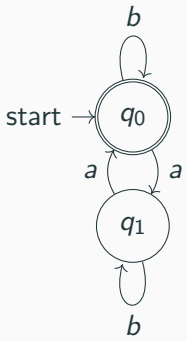
$p_1(a) = 0.3$ $p_2(a) = 1$ $p_3(a) = 0.5$ $p_4(a) = 0.3$
$p_1(b) = 0.7$ $p_2(b) = 0$ $p_3(b) = 0.5$ $p_4(b) = 0.7$

q_0	1	0.7
q_1	0	0.3

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:



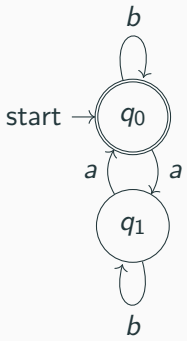
$p_1(a) = 0.3$ $p_2(a) = 1$ $p_3(a) = 0.5$ $p_4(a) = 0.3$
$p_1(b) = 0.7$ $p_2(b) = 0$ $p_3(b) = 0.5$ $p_4(b) = 0.7$

q_0	1	0.7	0.3
q_1	0	0.3	0.7

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:



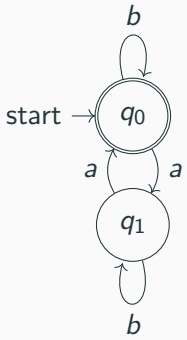
$p_1(a) = 0.3$ $p_2(a) = 1$ $p_3(a) = 0.5$ $p_4(a) = 0.3$
$p_1(b) = 0.7$ $p_2(b) = 0$ $p_3(b) = 0.5$ $p_4(b) = 0.7$

q_0	1	0.7	0.3	0.5
q_1	0	0.3	0.7	0.5

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:



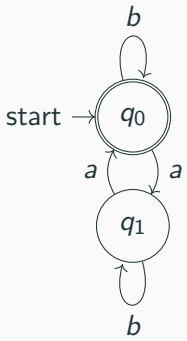
$p_1(a) = 0.3$ $p_2(a) = 1$ $p_3(a) = 0.5$ $p_4(a) = 0.3$
$p_1(b) = 0.7$ $p_2(b) = 0$ $p_3(b) = 0.5$ $p_4(b) = 0.7$

1	0.7	0.3	0.5	**0.5**
0	0.3	0.7	0.5	0.5

Tractability for Regular Languages

Folklore: For any fixed regular language L, the problem $\#\mathrm{pM}(L)$ is in PTIME.

Proof: Take $\mathcal{A}$ a DFA for L and do dynamic programming:



	$p_1(a) = 0.3$ $p_1(b) = 0.7$	$p_2(a) = 1$ $p_2(b) = 0$	$p_3(a) = 0.5$ $p_3(b) = 0.5$	$p_4(a) = 0.3$ $p_4(b) = 0.7$	
q_0	1	0.7	0.3	0.5	**0.5**
q_1	0	0.3	0.7	0.5	0.5

This even works for unambiguous automata (only one accepting run per word)

Tractability for UCFG (Result Statement)

We study $\#\mathrm{pM}(L)$ for **context-free languages** (CFL) L. Recall the definitions:

- A CFL is a language recognized by a **context-free grammar** (CFG):
  - Ex: $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ with the CFG G_1 having productions $S \to aSb$ and $S \to \varepsilon$
- A word is accepted by G if it has a **derivation tree**
- A CFG is **unambiguous** (UCFG) if every word has at most one derivation tree
- A CFL is **unambiguous** (UCFL) if it is accepted by a UCFG, otherwise it is **inherently ambiguous**
  - Example: L_1 is a UCFL

Tractability for UCFG (Result Statement)

We study $\#\mathrm{pM}(L)$ for **context-free languages** (CFL) L. Recall the definitions:

- A CFL is a language recognized by a **context-free grammar** (CFG):
  - Ex: $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ with the CFG G_1 having productions $S \to aSb$ and $S \to \varepsilon$
- A word is accepted by G if it has a **derivation tree**
- A CFG is **unambiguous** (UCFG) if every word has at most one derivation tree
- A CFL is **unambiguous** (UCFL) if it is accepted by a UCFG, otherwise it is **inherently ambiguous**
  - Example: L_1 is a UCFL

Theorem

For any UCFL L, the problem $\#\mathrm{pM}(L)$ is in PTIME

Tractability for UCFG (Proof Sketch)

- Recall the **CYK** algorithm to test if a word $w = a_1 \cdots a_n$ is accepted by a CFG G
  - Build a table of Booleans $T[A, i, j]$ indicating whether $a_i \cdots a_{j-1}$ is derivable from A
  - Inductive case: $T[A, i, j] = \bigvee_{A \to BC} \bigvee_{i \leq k \leq j} T[B, i, k] \wedge T[C, k, j]$

Tractability for UCFG (Proof Sketch)

- Recall the **CYK** algorithm to test if a word $w = a_1 \cdots a_n$ is accepted by a CFG G
  - Build a table of Booleans $T[A, i, j]$ indicating whether $a_i \cdots a_{j-1}$ is derivable from A
  - Inductive case: $T[A, i, j] = \bigvee_{A \to BC} \bigvee_{i \leq k \leq j} T[B, i, k] \wedge T[C, k, j]$

- Generalization on a probabilistic word $p = p_1 \cdots p_n$: build a table of rationals $P[A, i, j]$ indicating the probability that $p_i \cdots p_{j-1}$ generates a word derivable from A
- Inductive case: $P[A, i, j] = \sum_{A \to BC} \sum_{i \leq k \leq j} P[B, i, k] \times P[C, k, j]$
- Uses **independent AND** (disjoint positions) and **exclusive OR** (unambiguity)

Intractability for Some Inherently Ambiguous Context-Free Languages

Hardness for a Fixed Grammar (I)

Theorem

There exists a CFL L for which the problem #pM(L) is **#P-hard**

Hardness for a Fixed Grammar (I)

Theorem

There exists a CFL L for which the problem $\#pM(L)$ is **#P-hard**

- Reduction from the #P-hard problem **#P2DNF**:
  - **Input:** A positive 2-DNF formula ϕ, i.e., a disjunction of terms $x_i \wedge x_j$
  - **Output:** The number of valuations that satisfy ϕ

Hardness for a Fixed Grammar (I)

Theorem

There exists a CFL L for which the problem #pM(L) is **#P-hard**

- Reduction from the #P-hard problem **#P2DNF**:
  - **Input:** A positive 2-DNF formula ϕ, i.e., a disjunction of terms $x_i \wedge x_j$
  - **Output:** The number of valuations that satisfy ϕ

- The input words will be of the form: $p = u \# v_1^R \# \cdots \# v_m^R \#$ where
  - u is a uniform probabilistic word coding a valuation
  - v_i^R is (the mirror of) a non-probabilistic word coding the i-th clause

Hardness for a Fixed Grammar (II)

Example: $\phi = (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$

Input words of the form $p = u \# v_1^R \# v_2^R \#$ with:

- $u = u_1 u_2 u_3$ with $u_i(0) = u_i(1) = 1/2$ for each i
- $v_1 = 110$ and $v_2 = 011$
- A valuation u' of u satisfies v_1 if $u'[1] = 1$ and $u'[2] = 1$, i.e., $v_1 \leq_{\text{componentwise}} u$

Hardness for a Fixed Grammar (II)

Example: $\phi = (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$

Input words of the form $p = u \# v_1^R \# v_2^R \#$ with:

- $u = u_1 u_2 u_3$ with $u_i(0) = u_i(1) = 1/2$ for each i
- $v_1 = 110$ and $v_2 = 011$
- A valuation u' of u satisfies v_1 if $u'[1] = 1$ and $u'[2] = 1$, i.e., $v_1 \leq_{\text{componentwise}} u$

Thus: $\#\text{pM}(L_0)$ is #P-hard for the language L_0 over $\Sigma = \{\#, 0, 1\}$ defined as:

$$L_0 := \{\ u \# \Sigma^* \# v^R \# \Sigma^* \mid u, v \in \{0,1\}^* \text{ with } v \leq_{\text{componentwise}} u \}$$

L_0 can be recognized by a nondeterministic pushdown automaton

Other Hard Languages

1. **The union of 2 UCFGs can be #P-hard**
   - Encode the run of a polynomial-time nondeterministic Turing Machine:
     - One CFG accepts if an error occurs on an even step
     - The other if an error occurs on an odd step

Other Hard Languages

1. **The union of 2 UCFGs can be #P-hard**
   - Encode the run of a polynomial-time nondeterministic Turing Machine:
     - One CFG accepts if an error occurs on an even step
     - The other if an error occurs on an odd step
2. **Nondeterministic One-Counter Automata can be #P-hard**
   - Encode #P2DNF again
   - Code each variable x_i in unary as $\#a^i\#$ to store i in the counter

Other Hard Languages

1. **The union of 2 UCFGs can be #P-hard**
   - Encode the run of a polynomial-time nondeterministic Turing Machine:
     - One CFG accepts if an error occurs on an even step
     - The other if an error occurs on an odd step
2. **Nondeterministic One-Counter Automata can be #P-hard**
   - Encode #P2DNF again
   - Code each variable x_i in unary as $\#a^i\#$ to store i in the counter

$\Rightarrow$ Is $\#\mathrm{pM}(L)$ hard for all inherently ambiguous CFLs L?

Other Methods for Tractability

Poly-Slicewise-Unambiguous Languages

L is **poly-slicewise-unambiguous** if, for each length n, we can compute in PTIME a UCFG capturing $L \cap \Sigma^n$

Theorem

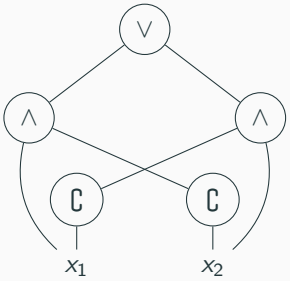
If L is poly-slicewise-unambiguous, then $\#\mathrm{pM}(L)$ is in PTIME

Consequences:

- #pM is tractable for **polyslender / bounded CFLs**
  - Polyslender = number of words of length n is polynomial in n
  - The UCFG for length n just generates each word explicitly
  - Covers some inherently ambiguous CFLs, e.g., $\{a^i b^j c^k \mid i = j \vee j = k\}$
- #pM is tractable for **unambiguous polynomial-time counter automata**
  - In particular: VASS, Parikh automata, one-counter automata, etc.
  - By contrast nondeterministic counter automata may be hard (previous slide)

Tractable Circuits

Boolean circuit C over variables $x_1, \ldots, x_n$: directed acyclic graph (DAG) with $\wedge$ / $\vee$ / C gates (AND, OR, negation).

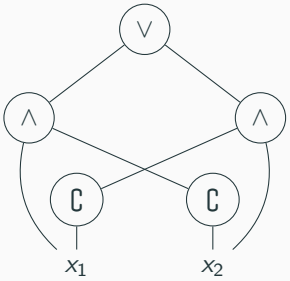


Tractable Circuits

Boolean circuit C over variables $x_1, \ldots, x_n$: directed acyclic graph (DAG) with $\wedge$ / $\vee$ / $\complement$ gates (AND, OR, negation).

C defines a language on $\Sigma = \{0, 1\}$ over words of length n:

$$L(C) = \{\, w \in \{0,1\}^n \mid C(w) = 1 \,\}.$$



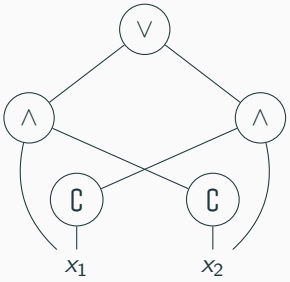
Tractable Circuits

Boolean circuit C over variables $x_1, \ldots, x_n$: directed acyclic graph (DAG) with $\wedge$ / $\vee$ / $\neg$ gates (AND, OR, negation).

C defines a language on $\Sigma = \{0, 1\}$ over words of length n:

$$L(C) = \{\, w \in \{0,1\}^n \mid C(w) = 1 \,\}.$$

$\#\mathrm{pM}(L(C))$ then amounts to computing the probability that C evaluates to true (but this may be #P-hard)



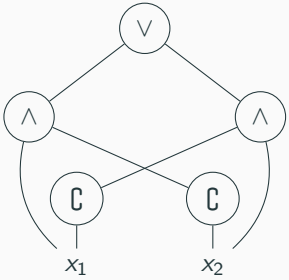
Tractability of Circuits

On which circuits C can we tractably compute the probability that C is true?

Tractable circuits

A circuit is **tractable** when:

- $\wedge$ gates are **decomposable**: no common variable in both inputs
- $\vee$ gates are **disjoint**: no common satisfying assignments



Tractability of Circuits

On which circuits C can we tractably compute the probability that C is true?

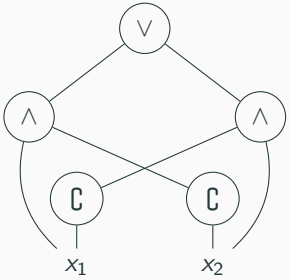
Tractable circuits

A circuit is **tractable** when:

- $\wedge$ gates are **decomposable**: no common variable in both inputs
- $\vee$ gates are **disjoint**: no common satisfying assignments

Theorem

Given such a circuit C on inputs $x_1, \ldots, x_n$ and a probabilistic word $p = (p_1, \ldots, p_n)$, we can solve $\#pM(L(C))$ in linear time (up to arithmetic costs)



Grammars vs Circuits

- Given a CFL L recognized by CFG G and length n we can build in PTIME a circuit C_n with $L(C^n) = L \cap \Sigma^n$
  - Proof: Trace of the CYK algorithm
- If the grammar is unambiguous, then we get a tractable circuit
- If L is poly-slicewise-unambiguous, then we can also build tractable circuits
- The converse is known to be false: $\{\Sigma^k a \Sigma^{n-1} a \Sigma^{n-1-k} \mid k \leq n-1\}$
  - $\rightarrow$ has tractable circuits, but not poly-slicewise-unambiguous
  - $\rightarrow$ see [Mengel and Vinall-Smeeth, PODS'25]

$\Rightarrow$ Intuition: tractable circuits allow an arbitrary reading order (vs left-to-right order for poly-slicewise-unambiguous)

Application to PAL^2

PAL^2 is the set of words of the form $w = w_1 w_2$ where w_1 and w_2 are palindromes

Theorem

$\#pM(\mathrm{PAL}^2)$ can be solved in $O(n^3)$ by building tractable circuits

Note that PAL^2 is an inherently ambiguous CFL (infinite ambiguity degree)

Application to PAL^2

PAL^2 is the set of words of the form $w = w_1 w_2$ where w_1 and w_2 are palindromes

Theorem

$\#pM(\mathrm{PAL}^2)$ can be solved in $O(n^3)$ by building tractable circuits

Note that PAL^2 is an inherently ambiguous CFL (infinite ambiguity degree)

Proof idea

- Key combinatorial lemma: The primitive root of any word in PAL^2 decomposes uniquely into two palindromes
- Then do a downward induction based on the **order** of the word
- Use subset difference to do inclusion-exclusion reasoning
- Same technique can show tractability of #pM for the language of primitive words

Conclusion and Future Work

Conclusion

- We studied probabilistic membership $\#\mathrm{pM}(L)$ for CFLs
- The problem is tractable for:
  - Unambiguous CFLs
  - Poly-slicewise-unambiguous languages, including polyslender languages
  - Unambiguous polynomial-time counter automata,
  - Languages with tractable $\times, \uplus, \mathbb{C}$-circuits, including primitive words and PAL^2
- Intractable already for:
  - Some unions of two UCFLs
  - Some nondeterministic counter automata

Conclusion and Future Work

Conclusion

- We studied probabilistic membership $\#\mathrm{pM}(L)$ for CFLs
- The problem is tractable for:
  - Unambiguous CFLs
  - Poly-slicewise-unambiguous languages, including polyslender languages
  - Unambiguous polynomial-time counter automata,
  - Languages with tractable $\times, \uplus, \mathbb{C}$-circuits, including primitive words and PAL^2
- Intractable already for:
  - Some unions of two UCFLs
  - Some nondeterministic counter automata

Future directions

- Classify more CFLs (e.g., PAL^k, PAL^*, etc.)
- Are tractable circuits with negation more powerful than without?
- Study combined complexity, with the automaton/CFG also given as input
- Complexity of uniform #pM? (all positions have the same distribution)
- Explore links with classical parsing complexity

Conclusion and Future Work

Conclusion

- We studied probabilistic membership $\#\mathrm{pM}(L)$ for CFLs
- The problem is tractable for:
  - Unambiguous CFLs
  - Poly-slicewise-unambiguous languages, including polyslender languages
  - Unambiguous polynomial-time counter automata,
  - Languages with tractable $\times, \uplus, \mathsf{C}$-circuits, including primitive words and PAL^2
- Intractable already for:
  - Some unions of two UCFLs
  - Some nondeterministic counter automata

Future directions

- Classify more CFLs (e.g., PAL^k, PAL^*, etc.)
- Are tractable circuits with negation more powerful than without?
- Study combined complexity, with the automaton/CFG also given as input
- Complexity of uniform #pM? (all positions have the same distribution)
- Explore links with classical parsing complexity

Thanks for your attention!

Tractability for Primitive Words (Result)

Definition
A word $w \in \Sigma^*$ is *primitive* if it is not a proper power:

$$w \neq u^k \quad \text{for any } u \in \Sigma^*,\ k \geq 2.$$

Let L_{prim} be the language of primitive words.

Theorem

The problem $\#\mathrm{pM}(L_{\mathrm{prim}})$ *can be solved in time* $O(n^2)$

Note that it is not known whether L_{prim} is context-free

Tractability for Primitive Words (Proof Sketch)

Root and order

Every $u \neq \varepsilon$ can be uniquely written as $u = w^d$ where w is primitive (the root) and $d \geq 1$ is the order.

Define:

$$L_k = \{\text{words of order } k\}, \qquad M_k = \{v^k \mid v \in \Sigma^*\}.$$

Note that M_k is the set of words whose order is a multiple of k.

Step 1: Build circuits for M_k (with $k \mid n$)

- Enforce periodicity: test that we have the same letter at each offset
- Product over offsets, union over letters, product over positions having that offset

Step 2: Recover $L_1 = L_{\text{prim}}$ by

$$M_k = \biguplus_{d \geq 1} L_{dk}$$

Downward induction, and subset difference via complementation: $V \setminus U = \complement((\complement V) \uplus U)$