16th International Workshop on Logic and Computational Complexity

(LCC 2026)

July 18-19, 2026, Lisbon

as part of the Federated Logic Conference FLoC'26

https://webusers.imj-prg.fr/~arnaud.durand/LCC/lcc26.html

Isabel Oitavem and Heribert Vollmer

(Chairs)

Abstract Booklet

Schedule

Day 1: Saturday July 18

Saturday morning 1: 09:00-10:40
Chair: Isabel Oitavem

- 09:00-10:00. B. Loff (invited talk): The Switching Lemma shows what the Switching Lemma cannot prove: The natural proofs barrier in constant-depth-circuits
- 10:00-10:40. V. Holzapfel: Recurrent Arithmetic Circuits

Coffee Break: 10:40-11:10

Saturday morning 2: 11:10-12:30
Chair: Heribert Vollmer

- 11:10-11:50. F. Chudigiewitsch, T. Tantau: On the Expressive Power of Modification Problems
- 11:50-12:30. F. Ferrari, E. Hainry, R. Péchoux, M. Silva: Quantum Programming in Polylogarithmic Time

Lunch: 12:30-14:00

Saturday afternoon 1: 14:00-15:20
Chair: Isabel Oitavem

- 14:00-14:40. K. Sauerwald, J. Kontinen, A. Meier: Complexity of Entailment for Cumulative Propositional Dependence Logics
- 14:40-15:20. L. Strieker: Uniformity in Transformer Models

Coffee Break: 15:20-15:50

Saturday afternoon 2: 15:50-17:10
Chair: Bruno Loff

- 15:50-16:30. A. Krapivin, B. Przybocki, B. Subercaseaux: The CNF Encoding Complexity of Boolean Functions and Non-Uniform Computation
- 16:30-17:10. K. Lange: Approximating the Values of Boolean Formulae in TC0

Day 2: Sunday, July 19

Sunday morning 1: 09:00-10:40
Chair: Isabel Oitavem

- 9:00-10:00. Arnold Beckmann (invited talk): On the Complexity of Confluence and Church-Rosser Proofs, with Applications to Bounded Arithmetic
- 10:00-10:40. M. D'Adda, G. Vanoni: Towards Higher-Order Logarithmic Space

Coffee Break 10:40-11:10

Sunday morning 2: 11:10-12:30
Chair: Arnold Beckmann

- 11:10-11:50. M. Ertel, E. Skapinakis: Characterizing levels of computational complexity by restrictions on hypothetical reasoning
- 11:50-12:30. J. Murwanashyaka: Notes on Sequential Theories

Lunch: 12:30 - 14:00

Sunday afternoon 1: 14:00-15:20
Chair: Heribert Vollmer

- 14:00-14:40. S. Lukumbuzya, M. Ortiz, M. Simkus: On the Expressive Power of Ontology-Mediated Queries: Capturing coNP
- 14:40-15:20. M. Bannach, J. Fichte, J. Groven, M. Hecher: Counting Complexity of ASP

Coffee Break: 15:20 - 15:50

Sunday afternoon 2: 15:50-17:00
Chair: Heribert Vollmer

- 15:50-16:50 Olaf Beyersdorff (invited talk): Proof complexity of QBF: relations to circuits and computationally hard problems
- 16:50-17:00 Closing

Complexity of Entailment for Cumulative Propositional Dependence Logics

Kai Sauerwald[1], Juha Kontinen[2] and Arne Meier[3]

[1]FernUniversität in Hagen, Hagen, Germany
[2]University of Helsinki, Helsinki, Finland
[3]Leibniz Universität Hannover, Hannover, Germany

Abstract

This paper establishes and proves complexity results for entailment for cumulative propositional dependence logic and for cumulative propositional logic with team semantics. As recently shown, cumulative logics are famously characterised by System C and exactly captured by the cumulative models of Kraus, Lehmann and Magidor. This gives rise to the entailment problem via relational models, which is specifically considered here.

Keywords

KLM, non-monotonic logic, System P, complexity, entailment, complexity, team logic, team semantics, preferential reasoning

1. Introduction

The ability to reason is one of the central features of intelligent agents and thus a major concern of artificial intelligence. In this paper, we study the fusion of non-monotonic reasoning with team-based reasoning. Such a combination is interesting, as it allows reasoning in which one can express settings involving a plurality of objects (the team-based component) while also taking into account extra-logical information, such as plausibility, exceptionality, reliability, preference, or typicality (the non-monotonic component). Notably, the combination of these approaches provides a setting that cannot be formalized by neither of the approaches alone.

Less is known about how to construct non-monotonic entailment relations for team-based logics. Known approaches for non-monotonic propositional logics or non-monotonic first-order logics rely heavily on the properties of the underlying logics, such as, e.g., the availability of all Boolean connectives, the law of excluded middle, and presence of material implication. However, these properties are not (always) available in team-based logics. There are many approaches to non-monotonic logics that take inspiration from [1], from which some are also generic, e.g., MAK models [2], KLM-style reasoning [3], characterization logics [4], or approximation fixpoint theory [5].

So far, there are only two pioneering works that consider a combination of team semantics and non-monotonic reasoning. First, the work by Yan (2023), which considers a non-monotonic team-based modal logic in the context of formal analysis of natural language. The second is by Sauerwald et al. (2025) (SMK), which also contains a broader introduction and motivation, and focuses on non-monotonic reasoning in the style of Kraus et al. (1990) (KLM).

KLM (1990) showed for classical propositional logic that cumulative entailment relations provide a stable theory of reasoning with multiple representations. Cumulative entailment relations are obtained by reasoning via the axiomatic system known as *System C*. Furthermore, a cumulative entailment relation $\mathrel{|\!\sim}$ can be represented by a cumulative model $\mathbb{C}$, by which $\varphi \mathrel{|\!\sim} \psi$ amounts (intuitively) to checking whether all minimal models of φ in $\mathbb{C}$ are models of ψ. SMK (2025) show that entailment via preferential models (which are specific cumulative models) satisfies *System C* in the context of propositional dependence logic. However, SMK provides no representation theorems for any kind of general cumulative reasoning in the context of team-based logics.

Contributions. This work considers the complexity of cumulative reasoning, which has been recently established as a stable approach in the context of team-based logics [8]. Specifically, we deal with the cumulative counterparts of the following team-based logics:

- Propositional logic with team-based semantics (TPL)
- Propositional dependence logic (PDL)

For each of these logics, we consider both reasoning via *System C* and via cumulative models as in the approach by KLM. This leads to *System C*-based entailment relations for PDL, denoted by $\mathsf{PDL}^{\mathsf{c}}$. Also, we will consider PDL entailment relations via cumulative models, denoted by $\mathsf{PDL}^{\mathsf{cuml}}$. With $\mathsf{TPL}^{\mathsf{cuml}}$ and $\mathsf{TPL}^{\mathsf{c}}$, we denote the respective approaches for TPL. We will study the complexity of cumulative entailment problems of these logics and obtain the following results:

1. Entailment for $\mathsf{PDL}^{\mathsf{cuml}}$ is in Θ_2^p and NP-hard, while it is in P and NC^1-hard for $\mathsf{TPL}^{\mathsf{cuml}}$.
2. Succinct Entailment for both logics is in Π_2^p and Δ_2^p-hard.

This underlines (under reasonable complexity assumptions) subtle differences in the complexities of these problems regarding the underlying logics.

2. Preliminaries

In this paper, we consider propositional logics from a model-theoretic perspective. We denote by $\mathsf{Prop} = \{ p_i \mid i \in \mathbb{N} \}$ the countably infinite set of propositional variables. We consider propositional formulas in negation normal form, i.e., PL-formulas are formed by the grammar, where $p \in \mathsf{Prop}$:

$$\varphi ::= p \mid \neg p \mid \bot \mid \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi.$$

We write $\mathsf{Prop}(\varphi)$ for the set of variables occurring in φ.

Classical Propositional Logic (CPL). For a non-empty finite subset $N \subseteq \mathsf{Prop}$ of propositional variables, one defines for valuations $v \colon N \to \{0,1\}$ over N and PL-formulas φ:

$$[\![\varphi]\!]^c = \{ v \colon N \to \{0,1\} \mid v \models \varphi \}.$$

16th International Workshop on Logic and Computational Complexity, July 18–19, 2026, Lisbon, Portugal

The valuation function v is extended to the *set of all* PL-*formulas satisfying* $\mathsf{Prop}(\varphi) \subseteq N$, $\mathrm{PL}(N)$, in the usual way. We denote by $\mathbb{A}_N$ the set of all assignments over N. We write $\varphi \models^c \psi$ for $[\![\varphi]\!]^c \subseteq [\![\psi]\!]^c$ and $\varphi \equiv^c \psi$ if both $\varphi \models^c \psi$ and $\psi \models^c \varphi$ are true.

Prop. Logic with Team Semantics (TPL)**.** Next, we define *team semantics* for PL-formulas (cf. [9, 10]). A *team* X is a set of valuations for some finite $N \subseteq \mathsf{Prop}$. The *domain* N *of* X is denoted by $\mathsf{dom}(X)$, and $\mathbb{T}_N$ denotes the set of all such teams.

Definition 1 (Team semantics of PL)**.** *Let X be a team. For any* PL*-formula φ with $\mathsf{dom}(X) \supseteq \mathsf{Prop}(\varphi)$, the* satisfaction relation*, $X \models \varphi$, is defined inductively as:*

$$
\begin{aligned}
&X \models p && \text{if for all } v \in X : v \models p,\\
&X \models \neg p && \text{if for all } v \in X : v \not\models p,\\
&X \models \top && \text{is always the case},\\
&X \models \bot && \text{if } X = \emptyset,\\
&X \models \varphi \wedge \psi && \text{if } X \models \varphi \text{ and } X \models \psi,\\
&X \models \varphi \vee \psi && \text{if there exist } Y, Z \subseteq X\\
& && \text{s.t. } X = Y \cup Z, Y \models \varphi, \text{ and } Z \models \psi.
\end{aligned}
$$

The set of all teams X with $X \models \varphi$ is denoted by $[\![\varphi]\!]^t$. For any two PL-formulas φ, ψ, we write $\varphi \models^t \psi$ if $[\![\varphi]\!]^t \subseteq [\![\psi]\!]^t$. Write $\varphi \equiv^t \psi$ if both $\varphi \models^t \psi$ and $\psi \models^t \varphi$ are true. We define the following properties for a formula φ:

$$
\begin{aligned}
&X \models \varphi \iff \text{for all } v \in X,\ \{v\} \models \varphi && \textbf{(Flatness)}\\
&\emptyset \models \varphi && \textbf{(Empty team)}\\
&X \models \varphi \text{ and } Y \subseteq X, \text{ then } Y \models \varphi && \textbf{(Downward closure)}
\end{aligned}
$$

Proposition 2. TPL *has the properties flatness, empty team, and downward closure.*

Due to the flatness property, logical entailment of propositional logic with team-based semantics $\models^t$ and classical semantics $\models^c$ coincide.

Propositional Dependence Logic (PDL)**.** A *(propositional) dependence atom* is a string $=(\vec{a}, b)$, in which $\vec{a} = a_1, \ldots, a_k$ and b are propositional variables from Prop. A team X *satisfies a dependence atom*, $X \models =(\vec{a}, b)$, if for all $v, v' \in X$, $v(\vec{a}) = v'(\vec{a})$ implies $v(b) = v'(b)$. A dependence atom where the first component is empty will be abbreviated as $=(p)$ and called a *constancy atom*. The language of *propositional dependence logic* (PL(dep)) is defined as PL-formulas extended by dependence atoms.

Example 3. *Consider the team X over $\{p, q, r\}$ defined by:*

	p	q	r
v_1	1	0	0
v_2	0	1	0
v_3	0	1	0

Here, we have that $X \models =(p, q)$ and $X \models =(r)$. Moreover, we have that $X \models =(p) \vee =(p)$, however it is true that $X \not\models =(p)$ as the value of p is not overall constant.

Proposition 4. PDL *has the empty team and the downward closure property, but not the flatness property.*

Generic View on Logics. Some parts of this paper require a generic perspective on logics, which we discuss next. A *satisfaction system* is a triple $\mathbb{S} = \langle \mathcal{L}, \Omega, \models \rangle$, where $\mathcal{L}$ is the set of *formulas*, Ω is the set of *interpretations*, and $\models \subseteq \Omega \times \mathcal{L}$ is the *satisfaction relation*. We write $[\![\varphi]\!]^{\mathbb{S}} = \{\omega \in \Omega \mid \omega \models \varphi\}$ for the set of all models of the formula $\alpha \in \mathcal{L}$. An *entailment relation* for a satisfaction system is a relation $\Vdash \subseteq \mathcal{L} \times \mathcal{L}$ such that $\alpha \Vdash \gamma$ if and only if $\beta \Vdash \gamma$, whenever $[\![\varphi]\!]^{\mathbb{S}} = [\![\psi]\!]^{\mathbb{S}}$. A satisfaction system $\mathbb{S}$ together with an entailment relation $\Vdash$ is called a *logic* and denoted by $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash \rangle$. The propositional logics discussed in this section fit into this general model-theoretic view for each $N \subseteq \mathsf{Prop}$ as follows:

$$
\begin{aligned}
\mathsf{CPL}_N &= \langle \mathrm{PL}(N), \mathbb{A}_N, \models, \models^c \rangle\\
\mathsf{TPL}_N &= \langle \mathrm{PL}(N), \mathbb{T}_N, \models, \models^t \rangle\\
\mathsf{PDL}_N &= \langle \mathrm{PL}(=(,)\,, N), \mathbb{T}_N, \models, \models^t \rangle
\end{aligned}
$$

When there is no ambiguity, we will write $\models$ instead of $\models^t$. Moreover, we will use CPL to denote the class consisting of all logics CPL_N for any non-empty finite subset $N \subseteq \mathsf{Prop}$. The classes TPL and PDL are defined analogous.

3. Cumulative Reasoning

We consider the construction of entailment relations.

***System C*.** We make use of the following rules for calculi:

$$
\frac{\varphi \equiv \psi \quad \varphi \mathrel{|\!\sim} \gamma}{\psi \mathrel{|\!\sim} \gamma}\ \text{(LLE)} \qquad \frac{\varphi \models \psi \quad \gamma \mathrel{|\!\sim} \varphi}{\gamma \mathrel{|\!\sim} \psi}\ \text{(RW)}
$$

$$
\frac{\varphi \wedge \psi \mathrel{|\!\sim} \gamma \quad \varphi \mathrel{|\!\sim} \psi}{\varphi \mathrel{|\!\sim} \gamma}\ \text{(Cut)} \qquad \frac{\varphi \mathrel{|\!\sim} \psi \quad \varphi \mathrel{|\!\sim} \gamma}{\varphi \wedge \psi \mathrel{|\!\sim} \gamma}\ \text{(CM)}
$$

Note that $\models$ is a placeholder for the entailment relation $\Vdash$ of an underlying logic $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash \rangle$, and $\equiv$ is the respective semantic equivalence from $\mathscr{L}$. *System C* consists of the rules (RW), (LLE), (CM), (Cut) and reflexivity (Ref) $\varphi \mathrel{|\!\sim} \varphi$ for all formulas [3, 11]. We say that an entailment relation $\mathrel{|\!\sim}$ *satisfies System C* if $\mathrel{|\!\sim}$ is closed under all rules of *System C*.

Relational Models. For a relation $R \subseteq \mathcal{S} \times \mathcal{S}$ on a set $\mathcal{S}$ and a subset $S \subseteq \mathcal{S}$, an element $s \in S$ is called *minimal in S with respect to R* if for each $s' \in S$ holds $\neg(s'\ R\ s)$. Then, $\min(S, R)$ is the set of all $s \in S$ that are minimal in S with respect to R. Moreover, for a set of interpretations M and a formula φ of $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash \rangle$, we write $M \models \varphi$ if for all $\omega \in M$ we have that $\omega \models \varphi$.

Definition 5 (Shoman 1988, Dix and Makinson 1989, 1992)**.** *Let $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash \rangle$ be a logic. A* relational model *for $\mathscr{L}$ is a triple $\mathbb{M} = \langle \mathcal{S}, \ell, R \rangle$ where $\mathcal{S}$ is a set, $\ell \colon \mathcal{S} \to \mathcal{P}(\Omega)$, and R is a binary relation on $\mathcal{S}$.*

We say $S \subseteq \mathcal{S}$ is *smooth* if for each $s \in S$, we either have that $s \in \min(S, R)$, or there exists a state $s' \in \min(S, R)$ with $s'\ R\ s$. For $\varphi \in \mathcal{L}$ and $s \in \mathcal{S}$, we denote by $\mathcal{S}(\varphi) = \{\, s \in \mathcal{S} \mid \ell(s) \models \varphi \,\}$ the set of states that satisfy φ. With $\min([\![\varphi]\!], R) = \bigcup\{\, \ell(s) \mid s \in \min(\mathcal{S}(\varphi), R) \,\}$, we denote the set of all interpretations that appear in $\ell(s)$ for any minimal state s that satisfy φ. We will deal, in this paper, with specific types of relational models, which we define in the following.

Definition 6. *A relational model $\mathbb{C} = \langle \mathcal{S}, \ell, R \rangle$ for a logic $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash \rangle$ is called* cumulative *if for all $\varphi \in \mathcal{L}$ the set $\mathcal{S}(\varphi)$ is smooth. A cumulative model $\mathbb{C}$ is* strong *if R is asymmetric (xRy implies $\neg(yRx)$) and $\min(\mathcal{S}(\varphi), R)$ has exactly one element for each formula $\varphi \in \mathcal{L}$.*

Problem	Tractable	Complexity	Result
ENT(PDL$^{\text{cuml}}$)	✗	$\in \Theta_2^p$, NP-hard	Thm. 8
SUCCENT(PDL$^{\text{cuml}}$)	✗	$\in \Pi_2^p$, Δ_2^p-hard	Thm. 10
ENT(TPL$^{\text{cuml}}$)	✓	$\in$ P, NC^1-hard	Col. 11
SUCCENT(TPL$^{\text{cuml}}$)	✗	$\in \Pi_2^p$, Δ_2^p-hard	Col. 11

Table 1
Overview of novel complexity results for entailment based on cumulative models for PDL and TPL. ENT/SUCCENT are the (succinct) entailment problem.

Because cumulative models satisfy smoothness, they avoid the problem of reasoning via arbitrary relational models, in which the set $\min(\llbracket\varphi\rrbracket, R)$ might be empty, even when $\mathcal{S}(\varphi)$ is non-empty. Entailment relations, which are induced by a relational model, are defined as follows.

Definition 7. *Let $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \Vdash\rangle$ be a logic. The* entailment relation $\mathrel{|\!\sim_{\mathbb{M}}} \subseteq \mathcal{L} \times \mathcal{L}$ *for a relational model $\mathbb{M} = \langle \mathcal{S}, \ell, R\rangle$ for $\mathscr{L}$ is given by*

$$\varphi \mathrel{|\!\sim_{\mathbb{M}}} \psi \text{ if } \min(\llbracket\varphi\rrbracket, R) \subseteq \llbracket\psi\rrbracket\,.$$

An entailment relation $\mathrel{|\!\sim} \subseteq \mathcal{L} \times \mathcal{L}$ is called *(strongly) cumulative* if there is a (strong) cumulative model $\mathbb{C}$ for $\mathscr{L}$ such that $\mathrel{|\!\sim} = \mathrel{|\!\sim_{\mathbb{C}}}$.

Classes of Entailment Relations. If L is a class of logics (such as PDL, TPL, or CPL), we denote with L$^{\text{c}}$ the class of all entailment relations $\mathrel{|\!\sim}$ for $\langle \mathcal{L}, \Omega, \models, \Vdash\rangle \in$ L that satisfy *System C*. Similarly, we use L$^{\text{cuml}}$ (L$^{\text{cuml[str]}}$) for the class of all (strong) cumulative entailment relations.

4. Complexity of Cumulative Entailments

In this section, we study the complexity of entailment for cumulative preferential logics. We consider both the standard and the succinct version of the problem, where the latter uses a circuit representation of the knowledge base. Recall, that a relational model $\mathbb{C} = \langle \mathcal{S}, \ell, R\rangle$ for a logic $\mathscr{L} = \langle \mathcal{L}, \Omega, \models, \models^{\mathscr{L}}\rangle$ is cumulative if for every formula $\varphi \in \mathcal{L}$ the set $\mathcal{S}(\varphi)$ is *smooth*, i.e., for each $s \in \mathcal{S}(\varphi)$ we have either $s \in \min(\mathcal{S}(\varphi), R)$ or there exists a state $s' \in \min(\mathcal{S}(\varphi), R)$ with $s'\ R\ s$. In the following, we will turn towards the central decision problems. We start with the standard entailment problem for cumulative propositional logic.

Problem:	ENT(PDL$^{\text{cuml}}$) — entailment problem for cumulative propositional dependence logic
Input:	A finite cumulative model $\mathbb{W} = \langle \mathcal{S}, \ell, \prec\rangle$ for PDL and $\varphi, \psi \in$ PL.
Question:	Is it true that $\varphi \mathrel{|\!\sim_{\mathbb{W}}} \psi$?

Theorem 8. ENT(PDL$^{\text{cuml}}$) *is in Θ_2^p and* NP*-hard.*

Proof. For membership, we make use of the cumulative input model $\mathbb{C} = \langle \mathcal{S}, \ell, R\rangle$. Regarding all $s \in \mathcal{S}(\varphi)$ we first need to compute $\mathcal{S}(\varphi)$ which is the set of all formulas such that $\ell(s) \models \varphi$. As the model checking problem for PDL is NP-complete [14, Thm. 1], we need to go through all states $s \in \mathcal{S}$ and check whether $\ell(s) \models \varphi$ holds. This can be done with an NP oracle. Next, we need to compute the minimal states $\min(\mathcal{S}(\varphi), R)$. This can be done by checking for each state $s \in \mathcal{S}(\varphi)$ whether there is a state $s' \in \mathcal{S}(\varphi)$ such that $s'\ R\ s$ holds. This can be done in polynomial time in the size of the model. To identify the minimal states, we only need to identify the sinks in the graph induced by R on $\mathcal{S}(\varphi)$. Finally, we need to check whether for all minimal states $s \in \min(\mathcal{S}(\varphi), R)$ it holds that $\ell(s) \models \psi$. Again, this can be done with an NP oracle. Thus, overall we can decide $\varphi \mathrel{|\!\sim_{\mathbb{W}}} \psi$ in Θ_2^p. Notice that this upper bound is in line with the upper bound for the preferential logic PDL$^{\text{pref}}$, which has to be considered as a special case of cumulative models.

The lower bound follows from the underlying classical complexity of model checking for PDL by constructing a simple cumulative model $(T, \varphi) \mapsto (\langle\{T\}, \mathrm{id}_{\mathcal{S}}, \emptyset\rangle, \top, \varphi)$ which is already preferential. □

For preferential models circuit families of size $(2^n)^{O(1)}$ for SUCCENT(PDL$^{\text{pref}}$) have been considered [7]. This was correct, since in such models, a state maps to a team. For cumulative models, a state maps to a set of teams, requiring a further exponential jump in the representation, i.e., we need to consider circuit families of size $(2^{2^n})^{O(1)}$ for SUCCENT(PDL$^{\text{cuml}}$). To that end, we will next define an adequate notion that appropriately addresses these large models.

Definition 9. *Let $N \subseteq$ Prop be a set of propositions with $|N| = n$, $\mathcal{S} = \{0,1\}^m$ be a set for $m \in (2^{2^n})^{O(1)}$, and $\prec\ \subseteq \mathcal{S} \times \mathcal{S}$ be a strict partial order. Now let $\mathbb{W}$ be a cumulative model $\mathbb{C} = \langle \mathcal{S}, \ell, \prec\rangle$ such that $\ell\colon \mathcal{S} \rightharpoonup \mathcal{P}(\mathbb{T}_N)$ is a partial labelling. Let there be two $n^{O(1)}$-sized circuit families $\mathcal{L}, \mathcal{O}$ (labelling, ordering) such that:*

1. *ℓ is computed by $\mathcal{L}$,*
2. *$\mathcal{O}\colon \mathcal{S} \times \mathcal{S} \rightharpoonup \{0,1\}$ is a partial function such that for $s, s' \in \mathcal{S}$, the circuit outputs 1 if and only if $s \prec s'$ is true.*

We call $(\mathcal{L}, \mathcal{O})$ an $n^{O(1)}$-sized circuit representation *of $\mathbb{W}$.*

We can derive from this definition the succinct entailment problem for cumulative propositional dependence logic, SUCCENT(PDL$^{\text{cuml}}$), which is the problem considering only instances that have a $n^{O(1)}$-sized circuit representation of the preferential model, i.e., the input then is of the form $\langle\langle \mathcal{O}, \mathcal{L}\rangle, \varphi, \psi\rangle$.

Theorem 10. SUCCENT(PDL$^{\text{cuml}}$) *is in Π_2^p and Δ_2^p-hard under $\leq_{\text{m}}^{\log}$-reductions.*

Entailment for cumulative preferential logics with team semantics. We derive the natural entailment problem for cumulative propositional logic with team semantics from the one for cumulative propositional dependence logic.

Problem:	ENT(TPL$^{\text{cuml}}$) — entailment problem for cumulative propositional logic with team-based semantics
Input:	A finite cumulative model $\mathbb{W} = \langle \mathcal{S}, \ell, \prec\rangle$ for TPL and $\varphi, \psi \in$ PL.
Question:	Is it true that $\varphi \mathrel{|\!\sim_{\mathbb{W}}} \psi$?

This results in a better complexity for the non-succinct version.

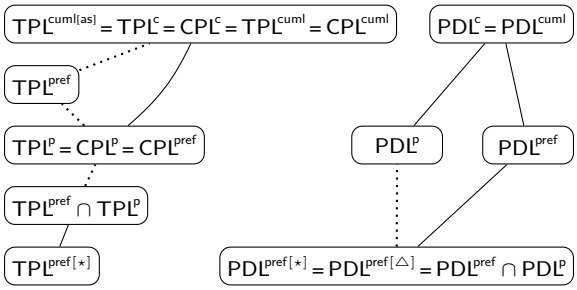


Figure 1: Landscape of classes of entailment relations. Solid lines denote strict inclusions, and dotted lines denote inclusions, where it is unknown whether the inclusion is strict. For not defined classes, consult SMK (2025).

Theorem 11. *The following holds:*

1. $\textsc{Ent}(\mathsf{TPL}^{\mathsf{cuml}}) \in \mathsf{P}$ *and is* NC^1*-hard.*
2. $\textsc{SuccEnt}(\mathsf{TPL}^{\mathsf{cuml}}) \in \Pi_2^p$ *and is* Δ_2^p*-hard under* $\leq_{\mathrm{m}}^{\log}$*-reductions.*

Proof. Regarding the first item, similarly as for preferential propositional logic with team semantics, the simpler model checking problem for the underlying team logic TPL—which is in P by [14, Tab. 1]—the oracle calls therefore become easier and can be computed on-the-fly in the P-machine. For the hardness, the lower bound from NC^1 (polynomial sized circuits of logarithmic depth with bounded fan-in; it is contained in nondeterministic logarithmic space) for classical propositional model checking [15] applies.

For item two, we can use $\forall\exists$-nondeterminism to check if there is no minimal state s that violates the entailment. That is, the label of s satisfies φ but not ψ. Regarding the lower bound, we can use the Δ_2^p-hardness of $\textsc{SuccEnt}(\mathsf{PDL}^{\mathsf{cuml}})$ and the fact that $\mathsf{TPL}^{\mathsf{pref}} \subseteq \mathsf{TPL}^{\mathsf{cuml}}$ to obtain a Δ_2^p-lower bound for $\textsc{SuccEnt}(\mathsf{TPL}^{\mathsf{cuml}})$. □

5. Conclusion

In this paper, we proved complexity results of entailment problems based on cumulative reasoning for propositional dependence logic and propositional logic with team semantics. These cumulative logics show a diverse landscape as shown in Figure 1 and now are better understood with respect to their computational complexity.

On our future agenda, we plan to find matching upper and lower complexity bounds yielding completeness results. Also logics, such as preferential reasoning (KLM 1990), c-inference [16], lexicographic inference [17] or System W [18], are interesting regarding complexity.

References

[1] G. Brewka, J. Dix, K. Konolige, Nonmonotonic Reasoning: An Overview, volume 73 of *CSLI Lecture Notes*, CSLI Publications, Stanford, CA, 1997.

[2] D. Makinson, General theory of cumulative inference, in: Proceedings of the 2nd International Workshop on Non-Monotonic Reasoning, Springer-Verlag, Berlin, Heidelberg, 1989, p. 1–18.

[3] S. Kraus, D. Lehmann, M. Magidor, Nonmonotonic reasoning, preferential models and cumulative logics, Artif. Intell. 44 (1990) 167–207. doi:10.1016/0004-3702(90)90101-5.

[4] R. Baumann, H. Strass, Consequence operators of characterization logics – the case of abstract argumentation, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning, Springer Nature Switzerland, Cham, 2025, pp. 154–166.

[5] M. Denecker, V. Marek, M. Truszczyński, Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning, Springer US, Boston, MA, 2000, pp. 127–144. doi:10.1007/978-1-4615-1567-8_6.

[6] J. Yan, Monotonicity in Intensional Contexts: Weakening and Pragmatic Effects under Modals and Attitudes, Ph.D. thesis, University of Amsterdam, 2023.

[7] K. Sauerwald, A. Meier, J. Kontinen, On the Complexity and Properties of Preferential Propositional Dependence Logic, in: Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, 2025, pp. 523–533. doi:10.24963/kr.2025/51.

[8] J. Kontinen, A. Meier, K. Sauerwald, On the complexity and properties of preferential propositional dependence logic, in: R. Wassermann, M.-L. Mugnier, F. Baader (Eds.), Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning, KR 2026, 2026.

[9] M. Hannula, J. Kontinen, J. Virtema, H. Vollmer, Complexity of propositional logics in team semantic, ACM Trans. Comput. Log. 19 (2018) 2:1–2:14.

[10] F. Yang, J. Väänänen, Propositional logics of dependence, Ann. Pure Appl. Logic 167 (2016) 557–589. doi:10.1016/j.apal.2016.03.003.

[11] D. M. Gabbay, Theoretical foundations for nonmonotonic reasoning in expert systems, in: K. R. Apt (Ed.), Logics and Models of Concurrent Systems, volume 13 of *NATO ASI Series*, Springer, 1984, pp. 439–457. doi:10.1007/978-3-642-82453-1_15.

[12] Y. Shoham, Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence, MIT Press, 1988.

[13] J. Dix, D. Makinson, The relationship between klm and mak models for nonmonotonic inference operations, Journal of Logic, Language, and Information 1 (1992) 131–140. URL: http://www.jstor.org/stable/40180016.

[14] J. Ebbing, P. Lohmann, Complexity of model checking for modal dependence logic, in: Proceedings of the 38th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2012), volume 7147 of *LNCS*, Springer, 2012, pp. 226–237.

[15] S. R. Buss, S. A. Cook, A. Gupta, V. Ramachandran, An optimal parallel algorithm for formula evaluation, SIAM J. Comput. 21 (1992) 755–780.

[16] G. Kern-Isberner, Conditionals in Nonmonotonic Reasoning and Belief Revision - Considering Conditionals as Agents, volume 2087 of *LNCS*, Springer, 2001.

[17] D. Lehmann, Another perspective on default reasoning, Ann. Math. Artif. Intell. 15 (1995) 61–82. doi:10.1007/BF01535841.

[18] C. Komo, C. Beierle, Nonmonotonic reasoning from conditional knowledge bases with system W, Ann. Math. Artif. Intell. 90 (2022) 107–144. doi:10.1007/S10472-021-09777-9.

The CNF Encoding Complexity of Boolean Functions and Non-Uniform Computation

Andrew Krapivin* , Benjamin Przybocki† , and Bernardo Subercaseaux‡

Carnegie Mellon University

May 2026

Given a boolean function f, how compactly can we "encode" f as a CNF formula? A partial answer was provided first by the Cook–Levin theorem [Coo71, Lev73], which implies that if f can be computed nondeterministically in n steps, then it can be encoded as a CNF instance of size $n^{O(1)}$. Since then, more compact encodings of size $\widetilde{O}(n)$ have been devised and extended to different models of computation [Sch78, PF79, Coo88, Rob91]. These optimized Cook–Levin theorems have several complexity-theoretic applications, including showing that CNF-SAT is complete for nondeterministic quasilinear time [Sch78, GS89] and proving time-space tradeoffs for CNF-SAT [For00]. The problem of compactly encoding boolean functions has also acquired tremendous practical utility with the advent of SAT solvers, where efficient encodings are often make-or-break for whether a problem can be solved in practice [SH23, KPSS25, HS24, QWSH25, HS15]; empirically, smaller encodings tend to perform better in practice [Bjö11, ANORC13]. Despite the theoretical and practical importance of CNF encodings, the minimum size of CNF encodings for different classes of boolean functions has yet to receive a systematic theoretical treatment, which is in stark contrast to other models of computation like boolean circuits [Juk12]. We initiate the study of CNF encodings from a theoretical lens, proving fundamental results about the complexity of boolean functions in this model and showing how this complexity metric relates to complexities in other non-uniform models. We believe that our results and proofs demonstrate that the theory of CNF encodings is much deeper than has previously been appreciated, and that it is fertile ground for the application of sophisticated combinatorial and algorithmic insights.

A CNF *encoding* of a boolean function f is a CNF formula with nondeterministic variables that computes f. Formally:

Definition 1 (Encoding)**.** *Given a boolean function $f\colon \{0,1\}^n \to \{0,1\}$, we say that a triple (φ, X, Y) is an* encoding *of f if: (i) $X := (x_1, \ldots, x_n)$ is a sequence of propositional variables, and Y is a set of propositional variables disjoint from X; (ii) φ is a CNF formula over variables $X \sqcup Y$; and (iii) for every assignment $\tau\colon X \to \{0,1\}$,*

$$f(\tau(x_1), \ldots, \tau(x_n)) = 1 \iff \exists \theta\colon Y \to \{0,1\},\ (\tau \cup \theta) \models \varphi.$$

The variables in X are called input *variables, and those in Y are called* auxiliary *variables.*

*andrew@krapivin.net
†benjamin.przybocki@gmail.com
‡bersub@cmu.edu

Thus, CNF encodings constitute a nondeterministic, non-uniform model of computation. We say that the *CNF encoding complexity* of a boolean function f, denoted $\mathsf{enc}(f)$ is the minimum number of clauses in a CNF encoding of f.

One of the few theoretical results known about encoding complexity is due to Tseitin [Tse70]: if the (nondeterministic) circuit complexity of f is n, then $\mathsf{enc}(f) = O(n)$. While Tseitin's transformation is well-known, and later refined by Plaisted and Greenbaum [PG86], there seems to have been no further work studying how $\mathsf{enc}(f)$ relates to other complexity measures. Each of our main results implies that the converse of Tseitin's theorem fails strongly: there are boolean functions for which $\mathsf{enc}(f)$ is much smaller than its (nondeterministic) circuit complexity. In fact, we have proven a circuit complexity characterization of $\mathsf{enc}(f)$ (up to a constant factor) in terms of the nondeterministic *conjunctive* circuit complexity of f. Thus, all of our results also have circuit complexity analogues.

Our results

Shannon kick-started the study of circuit complexity with his seminal result that the worst-case (nondeterministic) circuit complexity of n-bit boolean functions is $\Theta(2^n/n)$ [Sha49], which was later refined to $(1 + o(1))2^n/n$ by Lupanov [Lup58] and further tightened by Lozhkin [Loz22]. We prove an analogous result for encoding complexity. Let $\mathsf{enc}(n)$ be the maximum of $\mathsf{enc}(f)$ over all functions $f\colon \{0,1\}^n \to \{0,1\}$. We show that encoding complexity can be much smaller than circuit complexity in the worst case:

Theorem 2. *We have* $\mathsf{enc}(n) = \Theta\left(\sqrt{2^n}\right)$.

Our next result relates encoding complexity to the time complexity of nondeterministic RAM machines with advice. Robson showed the following variant of the Cook–Levin theorem: if $f\colon \{0,1\}^n \to \{0,1\}$ can be computed by a nondeterministic RAM machine in time $t \geqslant n$, then $\mathsf{enc}(f) = O(t \lg t)$. However, there is a precise sense in which Robson's result does not capture the full power of CNF encodings; nondeterministic RAM is a uniform model of computation, whereas CNF encodings constitute a nonuniform model. We show that functions that are efficiently computable by nondeterministic RAM machines *with advice* (a model we call 'ARAM') also have compact CNF encodings:

Theorem 3. *Let* $f\colon \{0,1\}^n \to \{0,1\}$*, and suppose that* f *can be evaluated in time* t *with* m *bits of advice by a nondeterministic* ARAM *machine that accesses at most* b *bits of the advice for each input. Then,* $\mathsf{enc}(f) = \widetilde{O}(\sqrt{mb} + t)$.

This theorem has a natural interpretation in terms of data structures. Suppose we have an algorithm to compute f that has access to a data structure D_f which uses m bits of memory, and upon receiving an input $\vec{x}$, nondeterministically computes $f(\vec{x})$ in time t by querying at most b bits of D_f. Then, Theorem 3 says that f must have a small CNF encoding, regardless of how long it takes to build D_f. This bound requires several nontrivial techniques to achieve. A CNF encoding without these techniques would be of size $\widetilde{O}(m + t)$, so our bound is quite powerful when m is large.

To demonstrate the power of Theorem 3, we give an application to encodings of k-CNF functions. The class of k-CNF functions is both theoretically natural and practically interesting, since many constraints are naturally formulated as k-CNF functions; for example, encoding that a subset of a graph forms an independent set is a 2-CNF function. It follows from results on partite decompositions of hypergraphs [CLT15, KPSMS25] that if $f\colon \{0,1\}^n \to \{0,1\}$ is k-CNF, then $\mathsf{enc}(f) = O_k(n^k/\lg n)$.[1] Furthermore, if we restrict our attention to encodings that are themselves

[1] The same bound holds for circuit complexity, where it is tight by an information-theoretic lower bound.

k-CNF (or even bounded width), then we have a matching $\Omega_k(n^k/\lg n)$ information-theoretic lower bound. We nevertheless show that $\mathsf{enc}(f) = o_k(n^k/\lg n)$, which rather surprisingly implies that clauses of unbounded width are helpful for encoding k-CNF functions:

Theorem 4. *If $f\colon \{0,1\}^n \to \{0,1\}$ is a k-CNF function for some $k \geqslant 2$, then $\mathsf{enc}(f) = \frac{n^k}{2^{\Omega_k(\sqrt{\lg n})}}$.*

We prove Theorem 4 by proving that given a k-CNF function f on n variables, there is a data structure using $O_k(n^k)$ bits of space that can evaluate f in time $\frac{n^k}{2^{\Omega_k(\sqrt{\lg n})}}$; together with Theorem 3, this immediately implies Theorem 4. Our data structural result is essentially a derandomization of a result of Larsen and Williams [LW17]. We believe that proving Theorem 4 without using Theorem 3 would be very difficult without new ideas.

We do not know how to obtain the encoding guaranteed by Theorem 4 with an efficient algorithm. It follows from our proof that the encoding can be constructed in exponential time. We do however have a (randomized) constructive alternative to Theorem 4, albeit at the cost of a weaker bound:

Theorem 5. *If $f\colon \{0,1\}^n \to \{0,1\}$ is a k-CNF function for some $k \geqslant 2$, then an encoding of f of size $o_k\left(\frac{n^k}{\lg(n)}\right)$ can be constructed with high probability by a randomized algorithm in polynomial time.*

The proof of this theorem does not use Theorem 3, so it has the advantage of being more explicit. The savings implied by the little-o term is polynomial in $\lg^*(n)$, a term that arises due to an invocation of Szemerédi's regularity lemma [Sze78]. We expect that the construction of Theorem 5 can be made deterministic, although we have not yet proven this.

References

[ANORC13] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 80–96. Springer, 2013. doi:10.1007/978-3-642-40627-0_9. 1

[Bjö11] Magnus Björk. Successful SAT Encoding Techniques. *Journal on Satisfiability, Boolean Modelling and Computation*, 7(4):189–201, 2011. URL: https://journals.sagepub.com/action/showAbstract, doi:10.3233/SAT190085. 1

[CLT15] László Csirmaz, Péter Ligeti, and Gábor Tardos. Erdős-Pyber theorem for hypergraphs and secret sharing. *Graphs Combin.*, 31(5):1335–1346, 2015. doi:10.1007/s00373-014-1448-7. 2

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047. 1

[Coo88] Stephen A Cook. Short propositional formulas represent nondeterministic computations. *Information Processing Letters*, 26(5):269–270, 1988. 1

[For00] Lance Fortnow. Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.*, 60(2):337–353, 2000. URL: https://doi.org/10.1006/jcss.1999.1671, doi:10.1006/JCSS.1999.1671. 1

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8, 1989, Proceedings*, Lecture Notes in Computer Science, pages 108–118. Springer, 1989. doi:10.1007/3-540-51237-3_10. 1

[HS15] Marijn J. H. Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Logic*, 16(3), June 2015. doi:10.1145/2736696. 1

[HS24] Marijn J. H. Heule and Manfred Scheucher. Happy ending: An empty hexagon in every 30 points. In *Tools and Algorithms for the Construction and Analysis of Systems: 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part I*, page 61–80, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-57246-3_5. 1

[Juk12] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4. 1

[KPSMS25] Andrew Krapivin, Benjamin Przybocki, Nicolás Sanhueza-Matamala, and Bernardo Subercaseaux. Optimal and efficient partite decompositions of hypergraphs, 2025. URL: https://arxiv.org/abs/2511.11855, arXiv:2511.11855. 2

[KPSS25] Markus Kirchweger, Tomáš Peitl, Bernardo Subercaseaux, and Stefan Szeider. From the finite to the infinite: Sharper asymptotic bounds on norin's conjecture via sat, 2025. URL: https://arxiv.org/abs/2511.08386, arXiv:2511.08386. 1

[Lev73] Leonid Levin. Universal'nyie perebornyie zadachi. *Problemy Peredachi Informatsii*, 9:265–266, 1973. 1

[Loz22] Sergei A. Lozhkin. Refined bounds on Shannon's function for complexity of circuits of functional elements. *Moscow Univ. Math. Bull.*, 77(3):144–153, 2022. Translation of Vestnik Moskov. Univ. Ser. I Mat. Mekh. **2022**, no. 3, 32–40. 2

[Lup58] Oleg B. Lupanov. On a method of circuit synthesis. *Izvestiya Vysshikh Uchebnykh Zavedeniy, Radiofizika*, 1:120–140, 1958. (in Russian). 2

[LW17] Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142. 3

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979. 1

[PG86] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986. URL: http://dx.doi.org/10.1016/S0747-7171(86)80028-1, doi:10.1016/s0747-7171(86)80028-1. 2

[QWSH25] Long Qian, Eric Wang, Bernardo Subercaseaux, and Marijn J. H. Heule. Unfolding boxes with local constraints. In *Automated Deduction – CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings*, page 736–754, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-3-031-99984-0_38. 1

[Rob91] John M Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82(1):141–149, 1991. 1

[Sch78] Claus-Peter Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM (JACM)*, 25(1):136–145, 1978. 1

[SH23] Bernardo Subercaseaux and Marijn J. H. Heule. The Packing Chromatic Number of the Infinite Square Grid is 15. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–406. Springer Nature Switzerland, 2023. 1

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.*, 28(1):59–98, 1949. URL: https://doi.org/10.1002/j.1538-7305.1949.tb03624.x, doi:10.1002/J.1538-7305.1949.TB03624.X. 2

[Sze78] Endre Szemerédi. Regular partitions of graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 399–401. CNRS, Paris, 1978. 3

[Tse70] Grigori S. Tseytin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, 1970. Translated from Russian (1968). 2

Recurrent Arithmetic Circuits

Vivian Holzapfel
Leibniz Universität Hannover
holzapfel@thi.uni-hannover.de

Generalising similar notions from the literature, we introduce the model of recurrent arithmetic circuits, which can be seen as arithmetic analogues of sequential or logical circuits. These circuits utilise so-called *memory gates* which are used to store data between iterations of the recurrent circuit. We introduce a notion of complexity classes for this model and show under which circumstances the newfound hierarchy of theses classes collapses.

Based on the model of a logical circuit mentioned by Wagner [3] we introduced the model of recurrent arithmetic circuits in [1] to study the complexity of recurrent Graph Neural Networks. The goal of this work is to further study the complexity of the introduced models and accompanying complexity classes. A similar model for Boolean circuits has been introduced Nickelsen et. al [2] where it was shown that its complexity is captured by PSPACE. In this work we are concerned with function problems over the reals $\mathbb{R}$, where the mentioned result cannot be directly reproduced. Instead we focus directly on a hierarchy in the complexity of recurrent arithmetic circuits.

General arithmetic circuits are a parallel model of computation and consist of a directed graph with gates that are labelled with addition and multiplication operations, as well as specific input and output gates. Given an initial input placed in the input gates they execute the operations until an output is computed. Classically the complexity of functions computable by arithmetic circuits has been studied in terms of the size (the number of non-input gates) and depth (the length of the longest path from an input to an output gate) of the circuits. To be able to talk about functions for each input size we define so called function families $(f_n)_{n\in\mathbb{N}}$ and their corresponding circuit families $(C_n)_{n\in\mathbb{N}}$, where for input size n the circuit C_n computes the function f_n.

The usual complexity classes associated are those from the FAC-hierarchy, i.e. $\mathrm{FAC}^i_{\mathbb{R}}$, consisting of function families computable by circuit families of size $O(n^{O(1)})$ and depth $O(\log^i n)$ where the gates have an unbounded in-degree and $\mathrm{FNC}^i_{\mathbb{R}}$ for bounded in-degree. It holds that $\mathrm{FAC}^i_{\mathbb{R}} \subseteq \mathrm{FNC}^{i+1}_{\mathbb{R}}$ for all $i \in \mathbb{N}$. These classical arithmetic circuits have a fixed depth and are only iterated once.

This model is now extended to that of a *recurrent arithmetic circuit* which allows multiple iterations of the computations of the circuit until a halting conditions defined by a halting function $f_{\text{halt}} \colon \mathbb{N} \times \mathbb{R}^{|V_{\text{halt}}|} \to \{0,1\}$ is met. This function takes the current iteration number and the

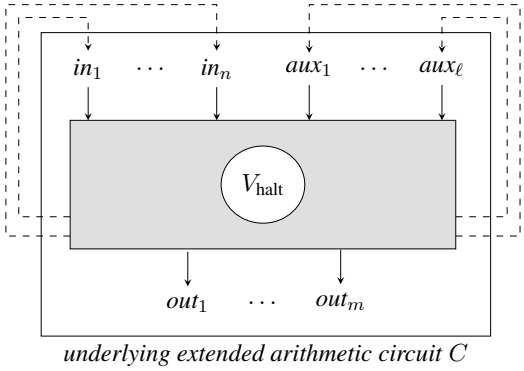


Figure 1: Structure of a recurrent arithmetic circuit: in_i are the input gates, aux_i are the auxiliary memory gates, out_i are the output gates and V_{halt} the halting gates. Dashed edges are the recurrent edges. The halting function is not depicted.

values of a subset V_{halt} of the gates into consideration and maps to 1 once the desired condition is fulfilled. We assume this function to be computable by an arithmetic circuit as well. Given the non-continuous nature of this function circuits computing it require an additional gate type *sign* which checks whether an input value to that gate is greater than zero. We write $\mathfrak{F}$ for an arbitrary FAC-hierarchy circuit function class and $\mathfrak{F}_s$ for the circuit function class with sign. By introducing so called *memory gates* we are able to retain information between the iterations of the circuit. Those memory gates consist of the input gates and additional auxiliary memory gates as needed. They have an incoming edge from the gate of which the value is to be saved for the next iteration. We refer to those as *recurrent edges*. We call the circuit without those recurrent edges that is then iterated the *underlying circuit*. An illustration of the model is given in Figure 1.

A recurrent arithmetic circuit computes a function as follows: In addition to the input gates, the auxiliary memory gates get assigned an initial constant. The underlying arithmetic circuit is then executed with the input values and some initial auxiliary memory gate values until all gates have a value. Afterwards the halting function is applied to the current iteration number and the values of the set of halting gates. If it evaluates to 1 the output of the function is the tuple of values in the output gates of the recurrent circuit. Otherwise the memory gates take the value of their respective predecessors, all other non-constant gates are reset to have no value and the computation of the underlying circuit is executed again.

We define recurrent circuit complexity classes by defining the complexity of the halting function and that of the function computed by the underlying circuit. We use $\text{rec}[\mathfrak{F}_s]$-$\mathfrak{F}'$ as a notation. Here the function of the underlying circuit is computed by a $\mathfrak{F}'$-circuit family and the halting function by a $\mathfrak{F}_s$-circuit family. We only consider circuit families from the FAC-hierarchy.

It follows easily that recurrent circuits are strictly more powerful than non recurrent ones, i. e. we have $\mathfrak{F} \subsetneq \text{rec}[\mathfrak{F}_{\text{halt}}]$-$\mathfrak{F}$ for circuit function classes $\mathfrak{F}$, $\mathfrak{F}_{\text{halt}}$, as classical arithmetic circuits can only compute polynomials and recurrent arithmetic circuits are able to compute recursive functions, for example the Fibonacci numbers. Given the knowledge about relations between the circuit function classes in the classical sense this leads to the question if and how the recurrent circuit function classes relate to each other. Naturally we would assume that $\text{rec}[\mathfrak{F}_{\text{halt}}]$-$\mathfrak{F} \subseteq$

$\text{rec}[\mathfrak{F}'_{\text{halt}}]$-$\mathfrak{F}'$ holds for any $\mathfrak{F}_{\text{halt}} \subseteq \mathfrak{F}'_{\text{halt}}$ and $\mathfrak{F} \subseteq \mathfrak{F}'$, but we are able to prove the much stronger results that we can show that the complexity of the underlying circuit can always be reduced to $\text{FAC}^0_{\mathbb{R}}$, i. e. functions computable by circuit families of constant depth.

Theorem 1. *Let $\mathfrak{F}_s$ and $\mathfrak{F}$ be circuit function classes. It holds that*

$$\textit{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}' \subseteq \textit{rec}[\mathfrak{F}_s]\text{-FAC}^0_{\mathbb{R}}.$$

The result relies on the recurrence and the additional memory gates of the recurrent circuit model. The underlying circuit is essentially stretched out to have its layers in parallel next to each other. Recurrent edges and memory gates replace the connections between the layers. The computations are done iteratively where after d iterations, where d is the depth of the original circuit, the computations of the complete circuit have been simulated. The original halting function is extended to only be checked after multiples of d iterations, which does not increase the complexity.

We moreover conjecture that we can obtain a similar result for the complexity of the halting circuit, namely

$$\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}' \subseteq \text{rec}[\text{FAC}^0_{\mathbb{R}}]\text{-}\mathfrak{F}_{\text{new}},$$

for circuit function classes $\mathfrak{F}_s, \mathfrak{F}'$ where $\mathfrak{F}_{\text{new}} = \mathfrak{F}$, if $\mathfrak{F} \subseteq \mathfrak{F}'$ and $\mathfrak{F}_{\text{new}} = \mathfrak{F}'$ otherwise. As the halting function is a piecewise function in two pieces that are defined by polynomial inequalities the computations of those polynomials can be moved to the underlying circuit instead leaving only the inequality checks and conjunction of those results in the halting circuit. Those computations can be done in constant depth, hence reducing the complexity to $\text{FAC}^0_{\mathbb{R}}$. The complexity of the underlying circuit might be increased, but as we showed in Theorem 1 we can also reduce that to $\text{FAC}^0_{\mathbb{R}}$. This results in our overall conjecture that we have

$$\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}' \subseteq \text{rec}[\text{FAC}^0_{\mathbb{R}}]\text{-FAC}^0_{\mathbb{R}}$$

for all circuit function classes $\mathfrak{F}', \mathfrak{F}_s$.

Further work could involve the study of a hierarchy within the recurrence, i. e. investigating the effects of halting functions that itself are computed by recurrent circuits or adding another layer of iterating an already recurrent circuit multiple times based on a second halting function.

References

[1] Timon Barlag, Vivian Holzapfel, Laura Strieker, Jonni Virtema, and Heribert Vollmer. Recurrent graph neural networks and arithmetic circuits. *CoRR*, abs/2603.05140, 2026. Extended version with Appendix.

[2] Arfst Nickelsen, Till Tantau, and Lorenz Weizsäcker. Aggregates with component size one characterize polynomial space. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 28.

[3] Klaus W. Wagner. *Theoretische Informatik - eine kompakte Einführung (2. Aufl.)*. Springer, 2003.

On the Expressive Power of Ontology-Mediated Queries: Capturing coNP

Sanja Lukumbuzya, Magdalena Ortiz and Mantas Šimkus

TU Wien, Austria

Abstract

This work investigates the *expressive power* of ontology-mediated queries (OMQs) from a *descriptive complexity* perspective. We show that OMQ languages based on non-monotonic extension of one of the most expressive description logics cannot express all coNP-computable Boolean queries, despite being coNP-complete in data complexity. We then propose an extension of this language and show that it is expressive enough to precisely capture the class of all Boolean queries computable in coNP.

1. Introduction

One of the central reasoning tasks in the context of *Ontology-Based Data Access* is answering *ontology-mediated queries* (OMQs), which are database-like queries coupled with an ontology that captures the domain knowledge to be taken into account during query answering. Two components define an OMQ language: (i) the language of the database query (e.g., instance or conjunctive queries etc.) and (ii) the language of the ontology. Description logics (DLs) [1] are a diverse family of logics specifically designed to reason about ontologies. In DLs, domains are modeled using constants (*individuals*), unary predicates (*concept names*), and binary predicates (*role names*), as well as a DL-dependent set of concept and role constructors and axiom schemata that determines the expressiveness and computational properties of the logic. The DL community has made significant efforts to understand the complexity landscape of DL-based OMQ languages both in terms of *combined complexity* and *data complexity*. Specifically, for OMQs based on the standard version of expressive DLs, we often have coNP-completeness in data complexity: this is the complexity of checking whether a given tuple of individuals is an answer to an OMQ Q over some fact base $\mathcal{A}$, assuming that Q is fixed and only $\mathcal{A}$ varies (see, e.g., [2]). *Relative expressiveness* of OMQ languages has also received some attention, e.g., via the established translations of OMQs into variants of Datalog (see, e.g., [3, 4, 5]).

However, the *expressive power* of OMQ languages from a descriptive complexity [6] perspective has thus far received only limited attention. In this context, we ask whether a given OMQ language is powerful enough to express all queries computable within some time or space resources, i.e., belonging to a certain complexity class. It was shown in [3] that there are OMQ languages that capture certain subclasses of coNP related to *constraint satisfaction problems* (CSPs). More recently, a close connection between OMQ languages with the so-called *closed*

sanja.lukumbuzya@tuwien.ac.at (S. Lukumbuzya)

predicates and *surjective CSPs* was shown in [7]. We continue this line of work and summarize our results presented in [8] where we focus on finding a DL-based OMQ language that precisely captures the complexity class coNP.

2. Expressive DLs with Closed Predicates

The DL $\mathcal{ALC}$ is considered the basic expressive DL in which can build complex expressions using concept names and the following concept constructors: $\top$ and $\bot$, concept *negation*, *conjunction*, *disjunction*, as well as the *existential* or *universal restriction* with respect to a role name. *DL ontologies*, also called *TBoxes* are collections of allowed terminological axioms. In the case of $\mathcal{ALC}$, ontologies consist of concept inclusions of the form $C \sqsubseteq D$ that state that every instance of concept C must also be an instance of D. A DL *knowledge base* (KB) consists of an ontology and a fact base, also called an *ABox*. A *model* of a DL ontology is a first-order interpretation that satisfies all terminological axioms, while a model of a KB must additionally satisfy all the ABox facts. In this work, we consider DLs $\mathcal{ALCHOI}$ and $\mathcal{ALCHOIF}$ that extend $\mathcal{ALC}$ with role inverses, *nominals* that make it possible to refer to a specific individual in the ontological part, and role inclusions ($r \sqsubseteq s$), and in the case of $\mathcal{ALCHOIF}$, functionality assertions (func r).

Most standard DLs, in particular also the logics introduced above, are fragments of first-order logic (FOL). As a result, the OMQ languages based on these logics are *monotonic* meaning that the inferred information cannot be invalidated with the addition of new facts. Monotonic OMQ languages cannot capture coNP since there exist (rather trivial) *non-monotonic* queries computable in coNP (or even polynomial time). Here is a simple example of such a (non-monotonic) query: "Does the input ABox have an odd number of individuals?". This means that our desired OMQ language must be non-monotonic.

A prominent way of introducing non-monotonicity to DLs is to equip them with the possibility to specify which ontological predicates are considered *closed* [9, 10], that is, their extensions are fully specified by the data. This is in contrast to the usual open-world assumption of FOL (and DLs). More specifically, a DL ontology with closed predicates consists of a set of terminological axioms $\mathcal{T}$ as well as a set Σ of predicates that will be interpreted exactly as given in the data. This means that all axioms of $\mathcal{T}$ that would typically allow us to infer new information over the closed predicates are viewed as integrity constraints. Naturally, closed predicates affect reasoning. Fortunately, it was shown in [4] and [5, 11] that answering IQs mediated by $\mathcal{ALCHOI}$ and $\mathcal{ALCHOIF}$ ontologies with closed predicates remains coNP complete in data complexity, which makes these logics excellent jumping off points of our investigation.

3. Overview of the Main Contributions

An OMQ language that captures coNP must be able to express all (so-called *generic*) Boolean queries (GBQs) that are computable in coNP. Each GBQ q has signature Σ and a set of ABoxes over Σ (also called Σ-ABoxes) in which q is "true". A query"q is computable in coNP" if there is a non-deterministic Turing Machine (NTM) M_q that recognizes the language of strings representing Σ-ABoxes in which q is "false" in polynomial time.

Our first result is an inexpressibility result for OMQ languages based on $\mathcal{ALCHOI}$ with closed predicates whose query answering problem can be reduced to the KB inconsistency problem for $\mathcal{ALCHOI}$ with closed predicates. The result is formulated for *Boolean inconsistency queries*, which are queries that simply ask whether a given KB has no models.

Theorem 1. *Boolean inconsistency queries mediated by $\mathcal{ALCHOI}$ ontologies with closed predicates cannot express all coNP-computable GBQs.*

We show this by carefully analyzing the runtime complexity of the existing KB satisfiability algorithm in [4] and invoking the *Non-deterministic Time Hierarchy Theorem*.

Next, we present $\mathcal{ALCHOIF}^+$, which extends $\mathcal{ALCHOIF}$ with closed predicates as well as restricted forms of transitivity, complex role expressions, and *nominal schemata* [12]. The features were specifically designed to ensure that we can incorporate them into the mosaic-based algorithm for reasoning in $\mathcal{ALCHOIF}$ with closed predicates [5] without increasing complexity.

Theorem 2. *The KB satisfiability problem for $\mathcal{ALCHOIF}^+$ with closed predicates is NP-complete in data complexity, and NExpTime-complete in combined complexity. Boolean inconsistency queries mediated by $\mathcal{ALCHOIF}^+$ ontologies with closed predicates are coNP-complete in data complexity and co-NExpTime-complete in combined complexity.*

Finally, we are able to show that combination of Boolean inconsistency queries and $\mathcal{ALCHOIF}^+$ can express every coNP-computable GBQ q by showing that language is powerful enough to express all computations of the non-deterministic Turing machine M_q associated with q.

Theorem 3 (Main result)**.** *For every coNP-computable GBQ Q over Σ there is a Boolean inconsistency query Q' mediated by an $\mathcal{ALCHOIF}^+$ ontology with closed predicates such that for all Σ-ABoxes $\mathcal{A}$, the answer to Q over $\mathcal{A}$ is true if and only if the answer to Q' over $\mathcal{A}$ is true.*

4. Future Work

We have shown that OMQ languages based on standard, monotonic DLs are inadequate for capturing the class of GBQs over ABoxes that are computable in coNP. We have also shown that inconsistency queries mediated by $\mathcal{ALCHOI}$ ontologies with closed predicates are still not powerful enough to capture coNP and we proposed a new OMQ language based on an extension of $\mathcal{ALCHOIF}$ with closed predicates that is able to do so.

The inexpressibility result for $\mathcal{ALCHOI}$ with closed predicates was formulated for inconsistency queries but also applies to instance queries and some fragments of CQs, as long as the query answering problem can be reduced to checking inconsistency of $\mathcal{ALCHOI}$ KBs with closed predicates. We note that it is unclear whether the inexpressibility result generalizes to OMQs with full conjunctive queries. This is because our result relies on the upper bound of running time of a known algorithm for consistency of $\mathcal{ALCHOI}$ KBs with closed predicates. To the best of our knowledge, no suitable upper bounds on data complexity are known for full CQs over $\mathcal{ALCHOI}$ KBs with closed predicates. Furthermore, it is also unclear whether one can prove an inexpressibility result for $\mathcal{ALCHOIF}$ with closed predicates without the additional features. This is left as future work.

References

[1] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. URL: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97.

[2] M. Ortiz, D. Calvanese, T. Eiter, Data complexity of query answering in expressive description logics via tableaux, J. Autom. Reason. 41 (2008) 61–98. URL: https://doi.org/10.1007/s10817-008-9102-9. doi:10.1007/s10817-008-9102-9.

[3] M. Bienvenu, B. ten Cate, C. Lutz, F. Wolter, Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP, ACM Trans. on Database Systems 39 (2014) 33:1–33:44. URL: http://doi.acm.org/10.1145/2661643. doi:10.1145/2661643.

[4] S. Ahmetaj, M. Ortiz, M. Simkus, Polynomial rewritings from expressive description logics with closed predicates to variants of datalog, Artificial Intelligence 280 (2020) 103220. URL: https://doi.org/10.1016/j.artint.2019.103220. doi:10.1016/j.artint.2019.103220.

[5] T. Gogacz, S. Lukumbuzya, M. Ortiz, M. Simkus, Datalog rewritability and data complexity of ALCHOIF with closed predicates, in: D. Calvanese, E. Erdem, M. Thielscher (Eds.), Proc. of KR 2020, 2020, pp. 434–444. URL: https://doi.org/10.24963/kr.2020/44. doi:10.24963/kr.2020/44.

[6] N. Immerman, Descriptive complexity, Graduate texts in computer science, Springer, 1999. URL: https://doi.org/10.1007/978-1-4612-0539-5. doi:10.1007/978-1-4612-0539-5.

[7] C. Lutz, I. Seylan, F. Wolter, The data complexity of ontology-mediated queries with closed predicates, Logical Methods in Computer Science 15 (2019). URL: https://doi.org/10.23638/LMCS-15(3:23)2019. doi:10.23638/LMCS-15(3:23)2019.

[8] S. Lukumbuzya, M. Ortiz, M. Simkus, On the expressive power of ontology-mediated queries: Capturing conp, in: O. Kutz, C. Lutz, A. Ozaki (Eds.), Proc. of the 36th International Workshop on Description Logics (DL 2023), CEUR Workshop Proceedings, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3515/paper-17.pdf.

[9] I. Seylan, E. Franconi, J. de Bruijn, Effective query rewriting with ontologies over dboxes, in: Proc. of IJCAI 2009, 2009, pp. 923–925. URL: http://ijcai.org/papers09/Papers/IJCAI09-157.pdf.

[10] C. Lutz, I. Seylan, F. Wolter, Ontology-based data access with closed predicates is inherently intractable(sometimes), in: Proc. of IJCAI 2013, IJCAI/AAAI, 2013, pp. 1024–1030. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870.

[11] S. Lukumbuzya, M. Ortiz, M. Simkus, Datalog rewritability and data complexity of ALCHOIQ with closed predicates, Artif. Intell. 330 (2024) 104099. URL: https://doi.org/10.1016/j.artint.2024.104099. doi:10.1016/J.ARTINT.2024.104099.

[12] M. Krötzsch, F. Maier, A. Krisnadhi, P. Hitzler, A better uncle for OWL: nominal schemas for integrating rules and ontologies, in: S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, R. Kumar (Eds.), Proc. of the 20th International Conference on World Wide Web, WWW 2011, ACM, 2011, pp. 645–654. URL: https://doi.org/10.1145/1963405.1963496. doi:10.1145/1963405.1963496.

Uniformity in Transformer Models

Laura Strieker

Leibniz Universität Hannover, strieker@thi.uni-hannover.de

1 Abstract

The notion of uniformity originates in circuit complexity [1], where it plays a central role. A single circuit C can process inputs of only one length; for example, C_5 handles inputs of exactly length 5. To process inputs of arbitrary length, one considers a circuit family $\{C_n\}_{n\in\mathbb{N}}$, where each C_n handles inputs of length n. While each circuit in the family computes the same underlying function restricted to its input length, in the general (non-uniform) case the individual constructions may differ arbitrarily. This leads to a key concern: even if the function computed by the family is computable, the description of C_n itself might be incomputable. Uniformity constraints address this by bounding the complexity of obtaining a description of C_n from n. Common notions include logspace-uniformity (L-uniform) and polynomial-time uniformity (P-uniform), which require that there exists an algorithm constructing C_n from input n in logarithmic space or in polynomial time, respectively [6]. These notions enable meaningful comparisons with machine models such as Turing machines, which natively handle inputs of arbitrary length.

Transformers occupy a special position in this landscape. Theoretically, they can process inputs of variable length; in practice, however, implementations typically impose a maximum input length N. Moreover, many complexity results for transformer models assume parameters that scale with the input length, such as internal numeric precision growing logarithmically with n. In addition, the transformer architecture admits a broad range of design and training parameters (e.g., number of layers, heads, precision, positional encodings), each of which can be coupled to n or to the maximum input length N.

These modeling choices have produced a rich but sometimes seemingly contradictory body of results on the computational complexity of transformers [5]. For instance, transformers have been shown to be Turing-complete under certain assumptions [4], yet standard formulations fail to recognize the TC^0-language PARITY [3]; at the same time, alternative formulations can solve PARITY [2]. These outcomes arise from different instantiations of the transformer model with varying assumptions about how architectural and numerical parameters depend on the input length n or on the maximum length N. Much work on transformer expressivity has followed in the formal language theory tradition of defining a problem as expressible if there exists a fixed transformer (with respect to the input length n) that can recognize a language. However, some work allows the hyperparameters of the network (e.g. depth, width) or even the parameter values themselves to depend on n.

A principled way to reconcile these findings is to explicitly parameterize the transformer model and introduce a corresponding notion of uniformity over its parameters. Unlike the classical circuit setting, where one constructs a distinct circuit C_n for each input length n, practical transformer deployments typically fix an architecture and specify parameter settings for all input lengths up to a maximum N. Consequently, the appropriate uniformity notion should characterize the complexity of obtaining, from n (and possibly N), the parameter values or schedules that realize the model's behavior at length n. Developing such a parameter-uniform framework would clarify assumptions across results and enable systematic comparisons between transformer-based computations and classical complexity classes.

1.1 A notion of uniformity

First, we need to fix the tuple of parameters that characterizes a transformer.

Definition 1. *A* transformer architecture $\mathcal{T}$ *is a Tuple* $(p, t, d, w, e, m, n, a, c)$ *where*

- p *denotes the precision,*
- t *temperature,*
- d *depth,*
- w *width,*
- e *positional encoding,*
- m *masking,*
- n *layer norm,*
- $a \in \{uha, aha, sma\}$ *attention,*
- c *number of chain of thought steps (or padding tokens).*

We call these the hyperparameters *of a transformer.*

The following example illustrates this definition.

Example 1. $\mathcal{T}_{\text{fu}} = (O(1), ..., O(1), \text{sma}, 0)$ are fixed-precision, temperature, and width transformers with softmax attention and no chain of thought. None of the hyperparameters depend on n (or N).

A transformer architecture $\mathcal{T}$ can be parameterized by a weight vector θ to obtain a language recognizer $\mathcal{T}_\theta : \Sigma^* \to \{0, 1\}$. We can then define uniform transformer classes in terms of these language recognizers and uniform families of parameters $\{\theta_N\}_{N=0}^{\infty}$:

Definition 2. *Let* $\mathcal{T}$ *be a transformer architecture. Then* U*-uniform-*$\mathcal{T}$ *is the set of languages* L *such that there exists a family of parameter vectors* $\{\theta_N\}_{N=0}^{\infty}$ *such that* $\mathcal{T}_{\theta_N}$ *recognizes* $L^{\leq N} = \{w \mid |w| \leq N \wedge w \in L\}$ *for all* N *and the function* $N \mapsto \theta_N$ *can be computed in the class* U.

Let $\mathcal{C}$ denote the set of constant functions $\mathbb{N} \to \Sigma^*$ and recall $\mathcal{T}_{fu}$. Then $\mathcal{C}$-uniform-$\mathcal{T}_{\text{fu}}$ is the expressive power of "fully uniform" transformers where the parameters cannot change at all with n. In contrast, $\mathcal{L}$-uniform-$\mathcal{T}_{fu}$ is the class where the number of parameters stays the same, but the values of θ can change uniformly in a way computable in log space. Finally, let $\mathcal{T}_w = (O(1), ..., O(1), O(\log n), sma, 0)$ denote transformers with logarithmic width. Then $\mathcal{L}$-uniform-$\mathcal{T}_w$ is the set of languages recognizable by transformers with logarithmic width, where the parameters can change with n in a way computable in logarithmic space. Thus, this single definition can be instantiated to make all of these notions of uniform transformer families precise.

An immediate next step is refining our understanding of separations between various types of transformers with different levels of uniformity, which could be relevant for understanding fundamental limits on length generalization: inexpressibility of a language under a fully uniform model implies length generalization is not possible on that language. More generally, better understanding the expressivity of uniform transformers can help us better conceptualize expressivity with a maximum context length and the role of parameters like depth or width in transformer expressivity.

References

1. Allan Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6(4):733–744, 1977.
2. David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7654–7664, Dublin, Ireland, May 2022. Association for Computational Linguistics.
3. Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguistics*, 8:156–171, 2020.
4. Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021.
5. Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Trans. Assoc. Comput. Linguistics*, 12:543–561, 2024.
6. Heribert Vollmer. *Relations to Other Computation Models*, pages 35–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

On the Expressive Power of Modification Problems

Florian Chudigiewitsch, Till Tantau
Universität zu Lübeck

April 28, 2026

Many important computational problems can be defined as *modification problems*. These problems provide an elegant framework for problem definitions and several Algorithmic Meta Theorems are known for them, providing a rich algorithmic toolbox. We study the expressive power of graph modification problems by providing non-expressibility proofs for particular problems and presenting foundational results that can be used as a basis for a more systematic study of the computational aspects of graph modification problems.

Fagin's Theorem [4] states that if C is a class of logical structures that is closed under taking isomorphisms, then C is in NP *if and only if* C can be defined by an existential second-order formula (ESO-formula).
A natural follow-up question is whether there are restrictions of ESO that can express all problems in NP. There are positive results, such as Leivant's Theorem [8], which states that every problem in NP can be defined by an ESO-formula that only uses universal quantifiers in the first-order part of the formula. On the other hand, it is known that existentially quantified binary predicates do not suffice to express all problems in NP, as shown by Ajtai [1]: If we consider structures consisting of a single m-ary relation, then the property "the number of m-tuples in the relation is even" cannot be expressed in existential second-order logic using any number of existentially quantified predicates whose arity is less than m. We also know that not every graph property decidable in NP can be defined in monadic existential second-order logic, a concrete example being graph connectivity [5].
A large class of problems that received considerable attention in research since the 1970s [12] are *modification problems*, where on an input, the task is to decide whether we can apply a restricted number of modifications so that the result satisfies certain conditions. As a running example, we will consider the *vertex deletion* meta-problem[1]. Let $\mathcal{P}$ be a fixed set of graphs.

Problem 1 (VERTEX-DELETION($\mathcal{P}$))**.**
Instance: *A graph* $G = (V, E)$*, a number* $k \in \mathbb{N}$.
Question: *Is there a set* $S \subseteq V$ *with* $|S| \leqslant k$ *such that* $G \setminus S \in \mathcal{P}$*?*

We abbreviate the problem as VD($\mathcal{P}$). Note that we have VD($\mathcal{P}$) $\in$ NP if $G \in \mathcal{P}$ is decidable in P. We write VD(FO) for the class of vertex deletion problems where the target class $\mathcal{P}$ is first-order definable, and VD($\mathtt{Q}_1\mathtt{Q}_2 \cdots \mathtt{Q}_c$) for $\mathtt{Q}_i \in \{\forall, \exists\}$ for the class of vertex deletion problems that are definable by an FO-formula of the form $\mathtt{Q}_1 x_1 \mathtt{Q}_2 x_2 \cdots \mathtt{Q}_c x_c \psi$, where ψ is quantifier-free. It is known that we can define NP-complete problems such as VERTEX-COVER as vertex deletion to first-order properties. We have:

$$\text{VERTEX-COVER} = \text{VD}(\forall x \forall y(\neg Exy)),$$

which shows that the class VD($\forall\forall$) already contains NP-complete problems, while one can also show that we have VD($\exists^*\forall\exists^*$) $\subseteq$ P, where "$\exists^*$" means that we allow arbitrary many existential quantifiers.
Recently, the *parameterized* complexity of the classes p-VD($\mathtt{Q}_1\mathtt{Q}_2 \cdots \mathtt{Q}_c$) was classified [2, 6], taking the number of vertices k we may delete as a parameter. For example, the works show that p-VD($\exists^*\forall^*\exists^*$) $\subseteq$

[1] We remark that many of the following considerations can also be applied to other graph modification operations such as edge deletion, edge flipping, elimination distance [7], or the meta-operation of *replacement* considered in [9].

FPT, while p-VD($\forall\exists\forall$) contains W[2]-hard problems. Thus, the results are *complexity-theoretic* insights from *logical* properties.
But this does not rule out that there are problems with a *lower* computational complexity, that we *cannot describe* as a problem in VD(FO) (or p-VD(FO)). To the best of our knowledge, definability questions in the style of ESO described above are not explored in the context of modification problems.
Intuitively, modification problems are connected to problems defined by logical formulas, as the to-be-modified parts of our input can also be "existentially guessed". We make this connection explicit later, and for now remark that the classifications above also show that the vertex deletion problems often have a lower complexity than their corresponding weighted definability problems (problems described by $\exists^{=}S\,\varphi$, where φ is a first-order formula) would suggest: All problems in p-VD($\exists^*\forall^*\exists^*$) are tractable and the problems in p-VD($\exists^*\forall^*$) are even in para-AC^0, the smallest class commonly considered in parameterized complexity. This suggests that the expressive power of vertex deletion problem also diverges from that of (weighted) ESO.

Non-Expressibility as Vertex Deletion Problems. Considering that we can describe NP-complete problems like vertex cover as a vertex deletion problem, a first question that arises from this is:

Can we describe every problem in NP *as a vertex deletion problem to a first-order formula?*

A simple observation casts serious doubt that this is the case: The ODD-CYCLE-TRANSVERSAL problem asks us to delete at most k vertices from an input graph such that the resulting graph has no cycles of odd length, or equivalently, is bipartite. But since bipartiteness is not expressible in first-order logic [10, Exercise 2.1.16], it seems unlikely that ODD-CYCLE-TRANSVERSAL is expressible as an vertex deletion problem where the target structure is first-order definable.
We also know that p-VD(FO) $\subseteq$ W[P], which means that under the common complexity-theoretic assumption that W[P] $\subsetneq$ para-NP, we cannot define any para-NP-hard problems with this framework. This is a nice example on how insights from *complexity theory* give us insights into inherently *logical* aspects of the formalism we study, like its expressive power.
But, as we will see, it is not always the descriptive complexity of the target set that can be a limiting factor, but that this also applies to the vertex deletion formalism as a whole: We show that there are problems that are not definable as vertex deletion problems, where that target set can be of arbitrary complexity.

Lemma 2. DOMINATING-SET *is not definable as a vertex deletion problem. That is, there is no set of graphs* $\mathcal{P}$ *such that we have* DOMINATING-SET $=$ VD($\mathcal{P}$).

Proof. Consider an instance $(G, 1)$ of dominating set, where G is a clique of size n. So we must also have $(G, 1) \in \text{VD}(\mathcal{P})$. We perform a case analysis by the vertices that can be deleted:

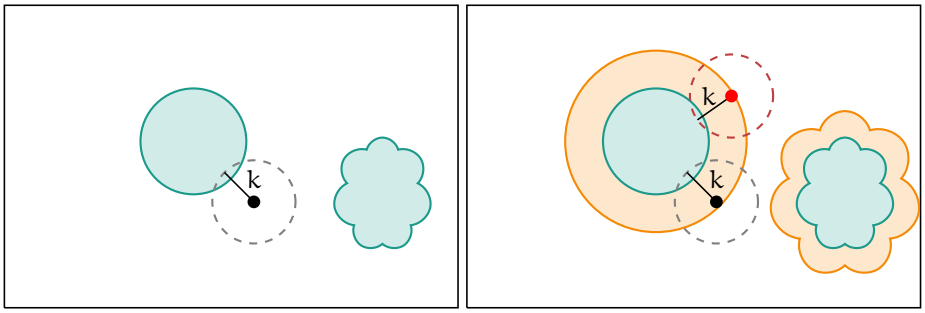
Case 1: No vertex is deleted from G. Then, we have $G \in \mathcal{P}$. But then we have $(G'', 1) \in \text{VD}(\mathcal{P})$, where G'' consists of a clique of size n and a single isolated vertex: We can delete the single isolated vertex and get $G \in \mathcal{P}$. But we have $(G'', 1) \notin$ DOMINATING-SET.

Case 2: A single vertex is deleted in G. Then, we have $K_{n-1} \in \mathcal{P}$. But then analogously we have $(G'', 1) \in \text{VD}(\mathcal{P})$ for $\mathcal{G}''$ a clique of size $n-1$ and a single isolated vertex, since we can again delete the isolated vertex to arrive at K_{n-1}, but, again, we have $(G'', 1) \notin$ DOMINATING-SET. □

A Necessary and Sufficient Condition for Definability as Modification Problems. We can generalize this observation in a very abstract manner. A function $d\colon \text{STRUC}[\tau] \times \text{STRUC}[\tau] \to \mathbb{N}$ is called a *distance measure* if for all $\mathcal{A}, \mathcal{B}, \mathcal{C} \in \text{STRUC}[\tau]$ we have

1. $d(\mathcal{A}, \mathcal{B}) = 0$ if and only if $\mathcal{A} = \mathcal{B}$, and
2. $d(\mathcal{A}, \mathcal{B}) \leqslant d(\mathcal{A}, \mathcal{C}) + d(\mathcal{C}, \mathcal{B})$.

Figure 1: Visualization of a modification problem on the left. The target set is drawn in green, with the instance being a dot and the radius of length k visualizing the allowed modification radius. On the right, a visualization of the idea of the proof on Proposition 3, where the orange area represents the distance from the target set that is small enough for the modification radius, and the red dot represents an instance that should not be in the modification problem.



Note that this definition bears some similarity to a metric, except that it is discrete and we do not demand symmetry. For a fixed distance measure $d\colon \text{STRUC}[\tau]\times\text{STRUC}[\tau] \to \mathbb{N}$ and a fixed set $\mathcal{P}$ of logical structures, a d-*modification problem with target set* $\mathcal{P}$ is then any problem that takes as input a logical structure $\mathcal{A}$ and a natural number k and asks whether there is a structure $\mathcal{B} \in \mathcal{P}$ with $d(\mathcal{A},\mathcal{B}) \leqslant k$.

Proposition 3. *Let* $Q \subseteq \text{STRUC}[\tau]$ *be a decision problem. Then, there exists a distance measure* d *and a set of logical structures* $\mathcal{P}$ *such that* Q' *is the* d*-modification problem with target set* $\mathcal{P}$ *and we have* $Q = Q'$ *if and only if we have that for all* $(\mathcal{A},k) \in Q$*, there exists an* $\mathcal{A}' \in \mathcal{P}$ *such that*

1. $d(\mathcal{A},\mathcal{A}') \leqslant k$ *and*
2. *for all* $\mathcal{A}''$*, we have that* $(\mathcal{A}'',\ell) \notin Q$ *implies* $d(\mathcal{A}'',\mathcal{A}') > \ell$.

This gives us an (albeit nonconstructive) method to prove that a problem is not definable as a modification problem. For this, we have to show that there exists an $(\mathcal{A},k) \in Q$ such that for all $\mathcal{A}' \in \mathcal{P}$, we have that $d(\mathcal{A},\mathcal{A}') \leqslant k$ implies that there exists an $(\mathcal{A}'',\ell) \notin Q$ with $d(\mathcal{A}'',\mathcal{A}') \leqslant \ell$.
We can furthermore show that a whole class of problems cannot be defined in terms of vertex deletion problems. A problem L that takes as input a logical structure and an integer is called *downwards closed* if $(\mathcal{A},k) \in L$ implies $(\mathcal{A},\ell) \in L$ for all $\ell \leqslant k$. A downwards closed problem L is called *non-trivial* if for all $\mathcal{A}$, there is an integer k with $(\mathcal{A},k) \notin L$.

Lemma 4. *Let* L *be a non-trivial downwards closed problem. Then, there is no distance measure* d *and a set of logical structures* $\mathcal{P}$ *such that* Q *is the* d*-modification problem with target set* $\mathcal{P}$ *and we have* $L = Q$.

Proof. Let L be a non-trivial downwards closed problem and for a distance measure d and a target set $\mathcal{P}$, let Q be a d-modification problem with target set $\mathcal{P}$ with $L = Q$. Observe that by definition, we must have that $\mathcal{P} = \bigcup_{(\mathcal{A},0)\in Q}\{\mathcal{A}\}$. Since L is non-trivial, we have furthermore that for every $\mathcal{A} \in \mathcal{P}$ that there is a $(\mathcal{A},\ell) \notin L$, but we have $(\mathcal{A},\ell) \in Q$, a contradiction. □

In particular, from this lemma it follows that many maximization problems that are central in the study of computational complexity like CLIQUE, INDEPENDENT-SET, and IRREDUNDANT-SET are not expressible as an vertex deletion problem.

Vertex Deletion as an ESO-Fragment. We can use the necessary and sufficient criteria for expressibility as a vertex deletion problem we have given above to show for individual problems. Now, we want to establish another view on vertex deletion problems, which will hopefully help us in building more

mathematical tools for showing expressibility. We again take the example of vertex deletion, and introduce a syntactically restricted fragment of ESO, wehre the target class is describable by a logic that has at most the expressive power of ESO. In the following, let $\mathcal{L}$ be one of the logics FO, FO[TC] (that is, first-order logic with a transitive closure operator), FO[LFP] (that is, first-order logic with a least-fixed-point operator), FO + conn (defined in [11]), or ESO.

Definition 5 (Relativization). *Let $\varphi = Q_1 x_1 \cdots Q_n x_n \psi(x_1, \dots, x_n)$ be a first-order τ-sentence where $Q \in \{\forall, \exists\}$, and ψ is quantifier-free. Let $\gamma(x)$ be a first-order τ-formula with one free variable. We define the* relativization $\varphi|_\gamma$ of φ with respect to γ *as*

$$\varphi|_\gamma := Q_1 x_1 \cdots Q_n x_n \left(\bigwedge_{i \in U} \gamma(x_i) \to \bigwedge_{i \in E} (\gamma(x_i) \wedge \psi(x_1, \dots, x_n)) \right),$$

where U (E, respectively) is the set of indices $i \in \{1, \dots, n\}$ such that Q_i is a universal (existential, respectively) quantifier.

Definition 6 (ESO_{vd}). *Let $\mathcal{L}$ be a logic and S a unary relation variable. The logic $\text{ESO}_{\text{vd}}(\mathcal{L})$ is the set of all formulas of the form*

$$\exists S \varphi|_{\neg S(x)},$$

where $\varphi \in \mathcal{L}$ and S may not occur positively in φ.

Lemma 7. $\text{ESO}_{\text{vd}}(\mathcal{L}) = \text{VD}(\mathcal{L})$.

In particular, we have $\text{ESO}_{\text{vd}}(\text{ESO}) \subseteq \text{ESO}$. Furthermore, we can characterize edge modification problems as fragments of relativized (guarded) second-order logic: Let S a binary relation variable. The logic for edge deletion $\text{ESO}_{\text{ed}}(\mathcal{L})$ is the set of all formulas of the form $\exists S \varphi|_{\neg S(x)}$, where $\varphi \in \mathcal{L}$ and S is a guarded variable (that is, we can quantify over edges) that may not occur positively in φ. Edge modification can be defined accordingly, where S is no longer guarded.

Further Research. We have given foundational results about the expressibility of modification problems, on the example of vertex deletion. We have seen that the dominating set problem is not expressible as a vertex deletion problem, stated necessary and sufficient criteria for definability as graph modification problems, and defined a fragment of existential second-order logic that captures modification problems.

Our first result was a strong statement for a single problem, while our second result was a weaker statement that was much more abstract. For more interesting results about the expressive power of modification problems, a "middle ground" between the two approaches would be the most promising. A possible candidate for this are the syntactic fragments of ESO we introduced in the last section.

An interesting avenue for further research would be the characterization of other modification operations as ESO-fragments, of which the elimination distance would be particularly interesting.

A tool that would make the fragment quite useful would be the definition of Ehrenfeucht-Fraïssé-style games for $\text{ESO}_T(\mathcal{L})$ for $T \in \{\text{vd}, \text{ed}, \text{em}\}$, which would provide more machinery to show non-expressibility.

Also, *extensional* ESO *logic* and *hereditary* FO *logic* considered in [3], which provides interesting results on expressibility, is tightly connected to vertex deletion. Study of these connections could provide more insights into the computational power of modification problems as well.

The logics for $(\mathcal{L})$ were chosen because they are popular in descriptive complexity and behave well with ESO. It would be interesting to extend the framework to more logics.

References

[1] Miklós Ajtai. "Σ^1_1-Formulae on finite structures". In: *Annals of Pure and Applied Logic* 24.1 (1983), pp. 1–48. DOI: 10.1016/0168-0072(83)90038-6.

[2] Max Bannach, Florian Chudigiewitsch, and Till Tantau. "On the Descriptive Complexity of Vertex Deletion Problems". In: *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024, August 26–30, 2024, Bratislava, Slovakia*. Ed. by Rastislav Královic and Antonín Kucera. Vol. 306. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 17:1–17:14. DOI: 10.4230/LIPICS.MFCS.2024.17.

[3] Manuel Bodirsky and Santiago Guzmán Pro. *On the Computational Power of Extensional ESO*. 2025. arXiv: 2511.08515 [math.LO]. URL: https://arxiv.org/abs/2511.08515.

[4] Ronald Fagin. "Generalized first-order spectra and polynomial-time recognizable sets". In: *Complexity of Computation* 7 (1974), pp. 43–74.

[5] Ronald Fagin. "Monadic generalized spectra". In: *Mathematical Logic Quarterly* 21.1 (1975), pp. 89–96. DOI: 10.1002/MALQ.19750210112.

[6] Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. "On the Parameterized Complexity of Graph Modification to First-Order Logic Properties". In: *Theory of Computing Systems* 64.2 (2020), pp. 251–271. DOI: 10.1007/s00224-019-09938-8.

[7] Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. "Parameterized Complexity of Elimination Distance to First-Order Logic Properties". In: *ACM Transactions on Computational Logic*. Vol. 23. 3. 2022, 17:1–17:35. DOI: 10.1145/3517129.

[8] Daniel Leivant. "Descriptive Characterizations of Computational Complexity". In: *Journal of Computer and System Sciences* 39.1 (1989), pp. 51–83. DOI: 10.1016/0022-0000(89)90019-6.

[9] Laure Morelle, Ignasi Sau, and Dimitrios M. Thilikos. "Graph Modification of Bounded Size to Minor-Closed Classes as Fast as Vertex Deletion". In: *33rd Annual European Symposium on Algorithms, ESA 2025, September 15–17, 2025, Warsaw, Poland*. Ed. by Anne Benoit et al. Vol. 351. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 7:1–7:18. DOI: 10.4230/LIPICS.ESA.2025.7.

[10] Martin Otto. *Finite Model Theory*. 2016.

[11] Nicole Schirrmacher, Sebastian Siebertz, and Alexandre Vigny. "First-order Logic with Connectivity Operators". In: *ACM Transactions on Computational Logic* 24.4 (2023), 30:1–30:23. DOI: 10.1145/3595922.

[12] Mihalis Yannakakis. "Node-and edge-deletion NP-complete problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. San Diego, California, USA: Association for Computing Machinery, 1978, pp. 253–264. ISBN: 9781450374378. DOI: 10.1145/800133.804355.

Quantum Programming in Polylogarithmic Time*

Florent Ferrari
ENS Lyon, France

Emmanuel Hainry,
Romain Péchoux,
Mário Silva
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

May 9, 2026

Abstract

Polylogarithmic time delineates a relevant notion of feasibility on several classical computational models such as Boolean circuits or parallel random access machines. As far as the quantum paradigm is concerned, this notion yields the complexity class FBQPOLYLOG of functions approximable in polylogarithmic time with a quantum random access Turing machine. We introduce a quantum programming language with first-order recursive procedures, which provides the first programming language-based characterization of FBQPOLYLOG. Each program computes a function in FBQPOLYLOG (soundness) and, conversely, each function of this complexity class is computed by a program (completeness). We also provide a compilation strategy from programs to uniform families of quantum circuits of polylogarithmic depth and polynomial size, whose set of computed functions is known as QNC, and recover the well-known separation result FBQPOLYLOG $\subsetneq$ QNC.

Introduction

Motivation. In order to check that quantum programs can be compiled and executed on a quantum computer, one has to design restrictions and constraints implying that quantum programs do not break the laws of quantum mechanics, for example, no-cloning of data and unitarity of operators. In addition, there is a need to tame their complexity in order to ensure their feasibility, that is, the fact that programs do not use too much space and time resources.

Taking inspiration from the classical world, this kind of issue has lead to the definition and study of several quantum polynomial time classes. One of the most striking examples of such classes is BQP, the quantum analog of the class of bounded-error probabilistic polynomial time problems BPP. By Yao's theorem [11], BQP corresponds exactly to what can be computed by uniform families of quantum circuits of polynomial size. This class, as well as its extension to functions, FBQP, have been characterized through various means, including function algebras [7] and first-order programs [5].

A natural question is then to study subpolynomial complexity classes. Polylogarithmic (polylog) time has been introduced and studied on Quantum Random-Access Turing Machine (QRATM) [4]. This definition uses random-access machines, as opposed to standard quantum Turing machines, because of the sub-linearity of time: although the machine cannot read its entire input, it can access any input bit or qubit. On quantum models, polylog time corresponds to problems that a QRATM can solve in a polylog number of steps, leading to the definition of the complexity class FBQPOLYLOG [8, 9] of functions computable with bounded-error in quantum polylog time. A main open problem is to design programming languages characterizing such a polylog class abstracting from low-level considerations [10].

Contributions. This paper makes a first step towards solving this problem. Towards that end, we introduce a quantum programming language with first-order recursive procedures, named PLP for PolyLog Programs (Figure 1), on which we obtain the following results:

- PLP programs are terminating and reversible.
- PLP is sound for FBQPOLYLOG, i.e., each PLP program computes a function in FBQPOLYLOG. Soundness relies on the use of a bounded recursion scheme for procedures to enforce the required polylog time properties, as illustrated by the binary search example.

*This is an extended abstract of [3] presented at MFCS'25.

(Integers)	i	$\triangleq$	$\mathrm{x} \mid k \mid \mathrm{i} \pm k \mid \mathrm{i}/2 \mid \lvert\mathrm{l}\rvert$
(Booleans)	b	$\triangleq$	$\mathrm{i} \geq \mathrm{i} \mid \ldots \mid \mathrm{b} \wedge \mathrm{b} \mid \ldots$
(Qubit lists)	l	$\triangleq$	$\bar{\mathrm{q}} \mid \mathrm{l} \ominus [\mathrm{i}] \mid \mathrm{l}^{\boxminus} \mid \mathrm{l}^{\boxplus}$
(Qubits)	q	$\triangleq$	l[i]
(Statements)	S	$\triangleq$	**skip**; $\mid$ q *= $\mathrm{U}^g(\mathrm{i})$; $\mid$ S S $\mid$ **if** b **then** {S} **else** {S} $\mid$ **qcase** q **of** $\{0 \to \mathrm{S}, 1 \to \mathrm{S}\}$ $\mid$ **call** $\mathrm{proc}(\mathrm{l}_1, \ldots, \mathrm{l}_n)$;
(Procedure decl.)	D	$\triangleq$	$\varepsilon \mid$ **decl** $\mathrm{proc}(\bar{\mathrm{q}}_1, \ldots, \bar{\mathrm{q}}_n)\{\mathrm{S}\}$, D
(Programs)	P	$\triangleq$	D :: S

Figure 1: Syntax of programs

- PLP is complete for FBQPOLYLOG, i.e., each function of this complexity class is computed by a PLP program. Completeness is shown by a direct encoding of polylog QRATM in PLP.
- PLP is also sound but not complete for QNC. For soundness, we outline a compilation algorithm that from a PLP program and its input size outputs a quantum circuit of polylog depth and polynomial size, i.e., circuits computing functions in QNC. Completeness does not hold as it is well-known that FBQPOLYLOG is strictly included in QNC.

Related Work. A characterization of BQPOLYLOG based on a function algebra has been provided in [8, 9], where a *fast quantum recursion scheme* is used to ensure that programs terminate in polylogarithmic time. Our work employs a similar bounded recursion scheme, using a simple divide-and-conquer strategy on qubits. This characterization can be seen as simpler and more natural approach since an imperative first-order programming language is easier to use.

First-Order Quantum Programs

Syntax. The considered language is a quantum programming language with first-order recursive procedures whose syntax is provided in Figure 1. There are four basic types for expressions: *Integer* expressions are variables x, constant $k \in \mathbb{N}$, arithmetic operations like $\mathrm{i} \pm k$ or $\mathrm{i}/2$, as well as the size $|\mathrm{l}|$ of a list of qubits l. *Boolean* expressions are defined in a standard way. *Qubit lists* are lists of unique qubit pointers. A qubit list expression l is either a variable $\bar{\mathrm{q}}$, the first (resp. second) half $\mathrm{l}^{\boxminus}$ (resp. $\mathrm{l}^{\boxplus}$) of the qubit list l, or a list $\mathrm{l} \ominus [\mathrm{i}]$ where the i-th element of l has been removed. *Qubit* expressions are of the shape l[i], which denotes the i-th qubit in l. We also define syntactic sugar for pointing to the n-th *last* qubit in a list, by defining for any $n \geq 1$, $\bar{\mathrm{q}}[-n] \triangleq \bar{\mathrm{q}}[|\bar{\mathrm{q}}| - n + 1]$.

A program $\mathrm{P} \triangleq \mathrm{D} :: \mathrm{S}$ is defined in Figure 1 as a list of procedure declarations D, followed by a program statement S. Statements include the no-op instruction, unitary application, sequences, conditional, quantum case, and procedure calls. For sake of universality [2], in a unitary application q *= U^g(i);, the unitary transformation $\mathrm{U}^g(\mathrm{i}) \in \mathbb{K}^{2\times 2}$ can take an integer i and a function $g \in \mathbb{Z} \to [0, 2\pi)$ as optional arguments[1], and we omit them when they are of no use.

The quantum conditional **qcase** q **of** $\{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$ allows branching by executing statements S_0 and S_1 in superposition according to the state of qubit q. We use a natural syntactic sugar when controlling on multiple qubits.

Semantics. Let $\mathbb{B} \triangleq \{0, 1\}$ denote the set of Booleans and $\mathcal{L}(\mathbb{N})$ denote the set of lists of natural numbers, [ ] being the empty list. We interpret basic types as follows: $[\![\text{Integers}]\!] \triangleq \mathbb{Z}$, $[\![\text{Booleans}]\!] \triangleq \mathbb{B}$, $[\![\text{Qubit lists}]\!] \triangleq \mathcal{L}(\mathbb{N})$, $[\![\text{Qubits}]\!] \triangleq \mathbb{N}$. Qubits are interpreted as integers (pointers) and qubit lists are interpreted as lists of pointers.

Given a program P, let the *length* of P be a function mapping each qubit variable $\bar{\mathrm{q}} \in Var(\mathrm{P})$ to an integer $\mathtt{len}(\bar{\mathrm{q}}) \in \mathbb{N}$. We write $\mathtt{len}_{\mathrm{P}}$ as a shorthand for $\sum_{\bar{\mathrm{q}} \in Var(\mathrm{P})} \mathtt{len}(\bar{\mathrm{q}})$ and $\mathtt{len}_{\mathrm{P}}^{<\bar{\mathrm{q}}}$ as a shorthand for

[1] In the case of quantum polynomial time, Adleman et al. [1] showed how the choice of amplitudes can affect the expressivity of classes such as BQP, requiring the restriction of *polynomial-time approximable complex amplitudes*. How the set of amplitudes influences the class FBQPOLYLOG remains an open question, as discussed in [8, 9], and so we abstain from the use of the entire set of complex numbers and instead use a field $\mathbb{K}$ which may refer to, for instance, polynomial-time approximable complex amplitudes.

SEARCH

```
1  decl search(q̄1, q̄2) {
2    if |q̄1| > 1 then {
3      qcase q̄1[|q̄1|/2, |q̄1|/2 + 1] of {
4        00 → call search(q̄1^⊞ ⊖ [1], q̄2); ,
5        01 → q̄2[1] *= NOT; ,
6        10 → call search(q̄1^⊟ ⊖ [−1], q̄2); ,
7        11 → skip; }
8    } else { skip; }
9  } ::
10 call search(q̄1, q̄2);
```

Figure 2: Binary search as an example of a PLP program.

$\sum_{\bar{q}' \in Var(P),\ \bar{q}' < \bar{q}} \texttt{len}(\bar{q}')$. A *configuration* c of program P over $\texttt{len}_P$ qubits is of the shape $(S, |\psi\rangle, A, f)$ with type $(\text{Statements} \cup \{\top, \bot\}) \times \mathcal{H}_{2^{\texttt{len}_P}} \times (Var(P) \to \mathcal{P}(\mathbb{N})) \times (Var(P) \to \mathcal{L}(\mathbb{N}))$, where $\top$ and $\bot$ are two special symbols denoting termination and error, respectively, where $|\psi\rangle \in \mathcal{H}_{2^{\texttt{len}_P}}$ is a quantum state, and where, for each qubit list $\bar{q} \in Var(P)$, $A(\bar{q})$ is the set of qubit pointers accessible from $\bar{q}$ and $f(\bar{q})$ is the list of qubit pointers assigned to $\bar{q}$.

Given a program $P \triangleq D :: S$, with $n = \texttt{len}_P$, let Conf_n be the set of configurations over n qubits. The initial configuration in Conf_n on input state $|\psi\rangle \in \mathcal{H}_{2^n}$ is $c_{init}(|\psi\rangle) \triangleq (S, |\psi\rangle, \bar{q} \mapsto \{1, \ldots, \texttt{len}(\bar{q})\}, \bar{q} \mapsto [1, \ldots, \texttt{len}(\bar{q})])$. A final configuration can be defined in the same way as $c_{final}(|\psi\rangle) \triangleq (\top, |\psi\rangle, \bar{q} \mapsto \{1, \ldots, \texttt{len}(\bar{q})\}, \bar{q} \mapsto [1, \ldots, \texttt{len}(\bar{q})])$. Each unitary transformation U of a unitary application q *= U^g(i); comes with a function $[\![U]\!]$ assigning a unitary matrix $[\![U]\!](g)(n) \in \mathbb{K}^{2\times 2}$ to each integer n and function $g \in \mathbb{Z} \to [0, 2\pi)$. We restrict ourselves to three kinds of gates: the phase gate Ph, rotation gate R_Y and NOT gate NOT, whose semantics is defined as follows:

$$[\![\text{Ph}]\!](g)(n) \triangleq \begin{pmatrix} 1 & 0 \\ 0 & e^{ig(n)} \end{pmatrix} \quad [\![R_Y]\!](g)(n) \triangleq \begin{pmatrix} \cos(g(n)) & -\sin(g(n)) \\ \sin(g(n)) & \cos(g(n)) \end{pmatrix} \quad [\![\text{NOT}]\!](\cdot)(\cdot) \triangleq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The big-step semantics $\cdot \longrightarrow \cdot$ is defined as a relation in $\bigcup_{n\in\mathbb{N}} \text{Conf}_n \times \text{Conf}_n$. We write $[\![P]\!](|\psi\rangle) = |\psi'\rangle$, whenever $c_{init}(|\psi\rangle) \longrightarrow c_{final}(|\psi'\rangle)$ holds.When an input state is defined by different qubit lists, we denote them in subscript. For instance, for $x, y \in \{0,1\}^*$ and $m \in \mathbb{N}$, we have that $|x\rangle_{\bar{q}_1} |y\rangle_{\bar{q}_2}$ indicates state $|x\rangle$ given as input to qubit list $\bar{q}_1$, state $|y\rangle$ given as input to qubit list $\bar{q}_2$.

Example (Binary search)**.** *Let $x \in 0^*1^*2^*$ be a sorted string and $\hat{x}$ denote the encoding of x as a binary given by $\hat{0} \triangleq 00$, $\hat{1} \triangleq 01$, and $\hat{2} \triangleq 10$. Program* SEARCH *in Figure 2 computes the function $[\![\texttt{SEARCH}]\!](|\hat{x}\rangle_{\bar{q}_1} |0\rangle_{\bar{q}_2}) = |\hat{x}\rangle_{\bar{q}_1} |b\rangle_{\bar{q}_2}$, where $b \in \{0,1\}$ indicates whether x contains a 1 or not.*

Polylogarithmic time restrictions. We define restrictions that guarantee termination polylogarithmic time and that their total number of sequential procedure calls is bounded polylogarithmically. Towards that end, we define a relation between procedures to account for recursion. Given two procedures proc and proc′ appearing in P, let proc $\sim_P$ proc denote that proc, proc′ are mutually recursive.

Definition. *A program* P *is said to be* recursively halving*, denoted* $P \in \text{HALF}$*, if for each procedure* proc $\in$ P *and for each procedure call* **call** $\text{proc}'(l_1, \ldots, l_n);$ $\in S^{\text{proc}}$*, if* proc $\sim_P$ proc′ *then there are* $1 \le i \le n$ *and* l *such that* $l^{\boxminus}$ *or* $l^{\boxplus}$ *appears in* l_i*.*

This restriction can be viewed as a recursion scheme, which implies a polylogarithmic time bound on programs in HALF by ensuring that in every (mutually) recursive procedure call at least one of the input qubit lists is cut in half. Now we impose a further condition on the number of sequential (mutually) recursive procedure calls. We define the *width* of a program P in the following way.

Definition. *Given a program* P*, the* width *of a procedure* proc $\in$ P *is defined by* $\text{width}_P(\text{proc}) \triangleq w_P^{\text{proc}}(S^{\text{proc}})$ *where* $w_P^{\text{proc}}(S)$ *is defined inductively on statements as follows. For basic instructions,*

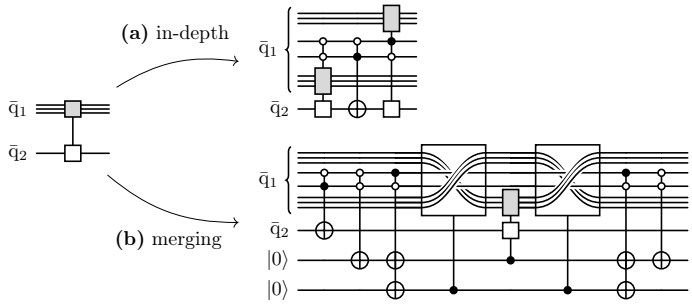


Figure 3: Compilation strategies for `search` defined in Figure 2.

$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{skip};) = w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{q}\ \texttt{*=}\ \mathrm{U}^g(\mathrm{i});) \triangleq 0$. *For the composition of statements,* $w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0\,\mathrm{S}_1) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0) + w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1)$. *Classical and quantum branching* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{if}\ \mathrm{b}\ \textbf{then}\ \{\mathrm{S}_1\}\ \textbf{else}\ \{\mathrm{S}_0\})$ *and* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{qcase}\ \mathrm{q}\ \textbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\})$ *both correspond to* $\max(w_{\mathrm{P}}^{\mathrm{proc}}(S_0), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1))$. *Procedure calls* $w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{call}\ \mathrm{proc}'(\mathrm{l}_1, \ldots, \mathrm{l}_n);) \triangleq 1$, *if* $\mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}', 0$, *otherwise. The* width *of a program* $\mathrm{width}(\mathrm{P})$ *is defined as* $\max_{\mathrm{proc} \in \mathrm{P}} \mathrm{width}_{\mathrm{P}}(\mathrm{proc})$. *Let* WIDTH$_{\leq 1}$ *be defined by* WIDTH$_{\leq 1} \triangleq \{\mathrm{P} \mid \mathrm{width}(\mathrm{P}) \leq 1\}$.

Definition. *The set* PLP *of* PolyLog Programs *is defined by* PLP $\triangleq$ HALF $\cap$ WIDTH$_{\leq 1}$.

Example. `SEARCH` $\in$ PLP. *This program only contains one procedure* `search`*, that is recursive. It is then easy to check that* `SEARCH` $\in$ HALF *as the recursive procedure calls are performed either on qubit lists* $\bar{\mathrm{q}}^{\boxminus}$ *or* $\bar{\mathrm{q}}^{\boxplus}$. *Furthermore, it is easy to verify that* $\mathrm{width}_{\texttt{SEARCH}}(\texttt{search}) = 1$.

A Characterization of Quantum Polylog Time

PLP characterizes exactly the functions in FBQPOLYLOG, the class of quantum polylog time approximable functions. That is, programs in PLP compute functions in FBQPOLYLOG and, reciprocally, for any function in FBQPOLYLOG, there exists a PLP program that computes it.

Main result. We denote by $[\![\text{PLP}]\!]$ the set of functions computed by programs in PLP. That is $[\![\text{PLP}]\!] \triangleq \{[\![\mathrm{P}]\!] \mid \mathrm{P} \in \text{PLP}\}$. A program P approximates function $f : \{0,1\}^* \to \{0,1\}^*$ with probability $p \in [0,1]$ if $\forall x \in \{0,1\}^*, |\langle f(x)|[\![\mathrm{P}]\!](x)\rangle|^2 \geq p$, in other words if for all input, the output of P coincides with f with probability at least p. The set of functions that can be approximated with probability at least p is denoted by $[\![\text{PLP}]\!]_{\geq p}$.

Theorem (Soundness & Completeness)**.** $[\![\text{PLP}]\!]_{\geq \frac{2}{3}} =$ FBQPOLYLOG.

Circuit compilation. The compilation strategy takes inspiration from [5, 6] which in particular uses ancillas to factorize the circuits representing procedure calls in branches so as to prevent exponential blow-up of the circuit size. Their technique is called *anchoring and merging* as when a procedure call is first encountered, an ancilla is associated to this call (*anchoring*), and when a subsequent call to this procedure happens with an input of the same size, this second call is then merged with the first (*merging*). Figure 3 exemplifies this on the `SEARCH` program: the left circuit represented by a grey and white box the circuit for the `search` procedure applied to $\bar{\mathrm{q}}_1, \bar{\mathrm{q}}_2$. Since this procedure has two calls to itself, its naive compilation gives an in-depth duplication of the calls. The anchoring/merging process entails a single recursive compilation at the price of an overhead in terms of ancillary qubits and permutations. Importantly, controlled permutations can be performed in logarithmic depth, which ensures that the all PLP programs are in QNC.

Theorem (Compilation)**.** *Given a* PLP *program* P*, and input size* $n = \sum_{\bar{\mathrm{q}} \in Var(\mathrm{P})} |\bar{\mathrm{q}}|$*, the quantum circuit produced by the compilation process is of size* $O(n\,\mathrm{polylog}(n))$ *and depth* $O(\mathrm{polylog}(n))$.

References

[1] Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997.

[2] P. Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing, 1999.

[3] Florent Ferrari, Emmanuel Hainry, Romain Péchoux, and Mário Silva. Quantum Programming in Polylogarithmic Time. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:17, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[5] Emmanuel Hainry, Romain Péchoux, and Mário Silva. A programming language characterizing quantum polynomial time. In *Foundations of Software Science and Computation Structures*, pages 156–175. Springer, 2023.

[6] Emmanuel Hainry, Romain Péchoux, and Mário Silva. Branch Sequentialization in Quantum Polytime. In Maribel Fernández, editor, *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*, volume 337 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] Tomoyuki Yamakami. A schematic definition of quantum polynomial time computability. *J. Symb. Log.*, 85(4):1546–1587, 2020.

[8] Tomoyuki Yamakami. Expressing power of elementary quantum recursion schemes for quantum logarithmic-time computability. In *Logic, Language, Information, and Computation (WoLLIC 2022)*, pages 88–104. Springer, 2022.

[9] Tomoyuki Yamakami. Elementary quantum recursion schemes that capture quantum polylogarithmic-time computability of quantum functions. *Mathematical Structures in Computer Science*, 34(7):710–745, August 2024.

[10] Tomoyuki Yamakami. Quantum first-order logics that capture logarithmic-time/space quantum computability. In Ludovic Levy Patey, Elaine Pimentel, Lorenzo Galeotti, and Florin Manea, editors, *CiE 2024*, pages 311–323. Springer, 2024.

[11] Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 352–361, 1993.

Approximating the Values of Boolean Formulae in TC0

Klaus-Jörn Lange (University of Tübingen)

May 10, 2026

1 Abstract

At the lower end of the hierarchy of complexity classes we find, represented by the word problem of the Dyck language $\mathbb{D}_1$ and the boolean formula value problem, respectively, the complexity classes TC^0 and NC^1, which are still not known to coincide or to be different. Aim of this note is to "approximate" the boolean formula value problem by TC^0-computable predicates. This is done by using the string representation of trees instead of using pointers. This allows for the quantification along the nodes of a path which usually requires logspace-complete techniques. Now it could be tempting to believe the boolean formula value problem to be TC^0, Which would have surprising consequences like the collapse of the counting hierrchy and its equality with $PSPACE$! We introduce the TC^0-computable predicates Pos and Neg which share many symmetries and dualities of the boolean case and shed some light on this possibility.

2 Preliminaries

Let $\mathbb{D}_1$ be the restricted (or one-sided) Dyck language over the letters a and b (instead of (and)). The equation $\mathbb{D}_1 = (a\mathbb{D}_1 b)^*$ will be used in the following without notice. A word $w \in \{a,b\}^*$ of length n will be denoted $w = w_1 w_2 \cdots w_n$. The length of a word w will be denoted by $|w|$. For $1 \leq i \leq j \leq n$ the subword $w_i w_{i+1} \cdots w_j$ is denoted by $w[i,j]$. In a decomposition $w = xaybz \in \mathbb{D}_1$ such that $y \in \mathbb{D}_1$ the pair of positions of a and b, i.e.$(|x|+1, n-|z|)$ is called a *matching pair*. In this case we denote the position $n-|z|$ by $m(|x|+1)$. Observe that the mapping $m()$ is TC^0-computable.

In this way a Dyck word describes an unlabelled tree: the nodes are represended by matching pairs. A matching pair $(i, m(i))$ is a *successor* of a matching pair $(l, m(l))$,denoted by $i \in S(l)$, iff $l < i < m(l)$ (and hence $l < m(i) < m(l)$). If $i \in S(l)$ we denote by $hd(l,i)$ the *height difference* between l and i, i.e. the length of the path between the nodes represented by l and i. i is called a *direct successor*, denoted by $Dirsuc(l,i)$ iff in addition $w[l+1, i-1] \in \mathbb{D}_1$ and hence also $w[m(i)+1, m(l)-1] in \mathbb{D}_1$ hold, i.e. $hd(l,i) = 1$. A path in the tree coded

by a Dyck-word is then a sequence of positions $i_1 < i_2 < \cdots < i_k$ such that $Dirsuc(i_j, i_{j+1})$ for all j. A *leaf* in this tree ischaracterized by a position t such that $m(t) = t + 1$ (and $w_t = a$).

2.1 Complexity

We assume familiarity with basic complexity classes such as $\mathbb{P}, Pspace, \mathsf{NC}^1$, or TC^0. Background on these topics may be found in textbooks such as [3]. We also assume familiarity with context-free grammars and regular expressions. We denote the empty string by λ.

3 Hahn's Coding of BFVP into $\mathbb{D}_1$

Building on similar constructions from [4], Hahn was able to embed the boolean formula value problem into $\mathbb{D}_1$ [2]: the context-free grammar for $\mathbb{D}_1$ with the rules $S \to SS \mid aSb \mid \lambda$ shows that $\mathbb{D}_1$ is built from the empty word by the operations concatenation and embracing. Hahn's interesting idea is to interpret concatenation as conjunction and embracing as negation setting the value of the empty word as true. Thus, λ and $aabb$ evaluate to true, while ab and $abab$ evaluate to false. In this way the boolean formula value problem is in a very natural way coded into $\mathbb{D}_1$. A context-free grammar for the resulting two languages H_+ and H_- with nonterminals T and F is

$$T \to aFb \mid TT \mid \lambda \quad \text{and} \quad F \to aTb \mid TF \mid FT \mid FF$$

with axioms T and F, respectively. In this way the Dyck language $\mathbb{D}_1$ is divided into the two disjoint subsets $\mathbb{D}_1 = H_+ \uplus H_-$.

Of course, we have

$$w \in H_+ \Leftrightarrow awb \in H_- \quad \text{and} \quad w \in H_- \Leftrightarrow awb \in H_+$$

for $w \in \mathbb{D}_1$. The following relations between H_+ and H_- are trivial and will be used without notice in the following:

$$H_+ = (aH_-b)^* \quad \text{and} \quad H_- = \mathbb{D}_1 a H_+ b \mathbb{D}_1.$$

3.1 Representing the BFVP by Dyckwords

In the following, we assume allways w to be a dyckword

The following equation relates the sets $\mathbb{D}_1$ and H_+:

Lemma 3.1. $H_+ = (a\mathbb{D}_1 a \cup b\mathbb{D}_1 b)^* \cap \mathbb{D}_1$ *([4])*

As a consequence we will see the crucial importance, whether leafs are of even or of odd distance to the root of a tree. For an atomic Dyck word $w \in a\mathbb{D}_1 b$ of length $n := |w|$ define $L^+(w) := \{1 \le i < n \mid m(i) = i + 1 \wedge hd(1, i) \equiv_2 1\}$ to be set of all leafs of odd height in the tree represented by w. Dually, we set

$L^-(w) := \{1 \leq i < n \mid m(i) = i + 1 \wedge hd(1, i) \equiv_2 0\}$ to be set of all leafs of even height in the tree represented by w.

It is quite easy to see, that $L^+(w)$ is the union of all $L^-(w[j, m(j)])$ over all direct successors j of the root of the tree represented by w, i.e. $Dirsuc(1, j)$. And dually, L^- is the union of all L^+ of all direct successors of the root.

If E^+ and E^- are the set of all atomic Dyckwords which have only positive, resp. negative, leafs, and NE^+ and NE^- their complements, i.e. the sets of all of atomic dyck words which have some positive, resp. negative, leafs, it is easy to show the following containments:

Lemma 3.2. *$E^- \subset H_+ \subset NE^+$. and thus for the conplements of these sets $E^+ \subset H_- \subset NE^-$*

Thus: the emptiness of $L^+(w)$ implies $w \in H_-$ which in turn implies $L^-(w)$ is not empty and dually $L^-(w)$ is empty implies $w \in H_+$ which in turn implies the nonemptiness of $L^+(w)$.

In addition, we have $w \in E^+ \equiv awb \in E^-$ and $w \in E^- \equiv awb \in E^+$ for atomic Dyckwords $w \in aDb$.

It should be noted, that the the sets E^+, E^- and NE^+, NE^- are in TC^0.

On the other hand, the word problems of H_+ and H_- are NC^1-complete.

4 Alternating Paths

In the following we consider atomic Dyckwords $w \in a\mathbb{D}_1 b$ fo the form $w = ad_1...d_k b$ for some atomic Dyck words $d_i = ad_{i1}...d_{ik_i}b, k > 0$ and all $k_i > 0$, where the $d_{i_j} \in a\mathbb{D}_1 b$

Then obviously $w \in H_+ \Leftrightarrow \exists_i d_i \in H_-$ and $w \in H_- \Leftrightarrow \forall_i d_i \in H_+$. Let $1 \leq i < n := |w|$ be a position in W with $w_i = a$ and let $i \leq t < m(t) = t+1 < m(i)$ be a leaf in the tree rooted in i. Let forther on be $j_0 := i < j_1 < \cdots j_r := t$ be the path (of direct successors) leading from i to t. We call this path *alternating*, iff the boolean values of $w[j_0, m(j_0)], w[j_1, m(j_1)], \cdots w[j_r, m(j_r)]$ alternate. Thus the last value ist *false*, since $w[i_r, i_r + 1]$ is a leaf, and the first value of the word $w[i, m(i)]$ is *true* iff r is odd, i.e. if t is a positve leaf of the subtree rooted in i.

Then $w[i, m(i)] \in H_+$ if and only if there is an alternating path leading from i to some positive leaf. Dually, $w[i, m(i)] \in H_-$ if and only if for each direct successor j of i there is an alternating path leading from i through j to some negative leaf in the subtree below i. Inspired by this we introduce the concept of a fan-shaped subtree leading to the TC^0-computable predicates Pos and Neg, The idea is to modify the dual relation between true anf false in the following way: while the atomic Dyck word w is false iff all of its direct successors are true, we define w to fulfill Neg iff at least one of its direct successors fulfills Pos and simultaneously all direct successors are "weakly positive" in the sense, that they are members of NE^+. And we say that w fulfills Pos if there is a positive leaf t in w, such that along the path from the root to t alternatingly on every second step we require this weak positiveness for all direct successors. Intuitively, we

require the existence of fan-shaped subtree of positive leafs. Formally, we define for a position i in an atomic Dyckword w that

$$w \models Pos(i) :\equiv \exists_{t \in L^+(i)} \forall_{i<j<t;\Delta(i,j)\equiv_2 1} \forall_{k \in DS(j)} NE^+(k)$$

. And we define Neg by

$$w \models Neg(i) :\equiv m(i) = i+1 \vee (\exists_j dissuc(i,j) \wedge Pos(j))$$

Let us define the sets $Pos := \{w \in a\mathbb{D}_1 b \mid w \models Pos(1)\}$,
$Neg := \{w \in a\mathbb{D}_1 b \mid w \models Neg(1)\}, NotPos := \{w \in a\mathbb{D}_1 b \mid \neg(w \models Pos(1))\}$, and $NotNeg := \{w \in a\mathbb{D}_1 b \mid \neg(w \models Neg(1))\}$.

It is quite easy to show

Lemma 4.1. *$E^+ \subset NotPos \subset H_- \subset Neg \subset NE^-$ and*
$E^- \subset NotNeg \subset H_+ \subset Pos \subset NE^+$.

. This leads to the following relations: $w \in Pos \Rightarrow awb \in Neg$ and $awb \in Pos \Leftarrow w \in Neg$.

Open Question: under which conditions do the reversals hold? If these conditions would be TC^0-computable, H_+ (and H_-) would be in TC^0! Obeserve, that we do not have $aawbb \in Pos$ implies $w \in Pos$ but only that $w \in Pos$ implies $aawbb \in Pos$, allthough we have $w \in NE^+ \Leftrightarrow aawbb \in NE^+$ as well as $w \in NE^- \Leftrightarrow aawbb \in NE^-$.

References

[1] D.A. Barrington. Bounded-width polynomial-size branching programs can recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[2] M. Hahn. An algebraic perspective on visibly pushdown languages. Master's thesis, Fachbereich Mathematik, Eberhard Karls Univerisit

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, Reading Mass., 1979.

[4] K.-J. Lange. The Boolean Formula Value Problem as Formal Language. In *Languages Alive*, number 7300, pages 138–144. Springer, 2012.

Towards Higher-Order Logarithmic Space

Martino D'Adda[†] Gabriele Vanoni[‡]

[†]Università di Milano

[‡]Université Paris Cité

Abstract

In this talk we introduce the type-2 analogue of FLOGSPACE, defined through oracle Turing machines, answering a question raised by Kawamura [4]. We show that this class, dubbed *Easy Feasible Functionals* (EFF) is robust, being closed by both type-0 and type-1 composition.

Higher-order computation was born together with computer science. We could say that the λ-calculus, the standard way to model higher-order computation, was the first universal theory of functions. However, when the study of computability theory shifted towards the study of complexity, the interest in higher-order computation diminished, being relegated mostly to the theory of programming languages. In fact, complexity theory is typically *first-order*, investigating functions of type $\mathbb{N} \to \mathbb{N}$, using Turing machines (and variants thereof) as the base computational model [1].

A notable exception is the study of type-2 functionals, *i.e.* functions of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$, where many results are available and a clear definition of type-2 *polynomial time* computability is accepted by the community. This definition is *robust* as it can be either given using oracle Turing machines (OTM), or implicitly, *e.g.* via a function algebra in the style of Cobham. The resulting class is often called *basic feasible functionals* (BFF) [3].

In this talk, we will concentrate on the following natural question: how can we define type-2 *logarithmic space*? As in the first-order case, dealing with logarithmic space is harder than dealing with polynomial time. We concentrate on machine models, leaving the definition of an implicit characterization (which is arguably simpler) for future work. When dealing with logarithmic space, computational resources are very constrained and the design of the model is crucial. Indeed, the idea is that a LOGSPACE (or L) algorithm can only store a *constant* number of *pointers* to its immutable input. Designing type-2 LOGSPACE means understanding how to constrain oracle Turing machines to work in LOGSPACE, in a reasonable way. Indeed, we would like to satisfy some desiderata, in particular the closure of this class w.r.t. composition. Since we are in the type-2 world, this means verifying both type-0 and type-1 composition, as follows:

Definition 1 (Composition)**.** Let $\mathcal{C}$ be a class of type-2 functionals.

- $\mathcal{C}$ is closed under type-0 composition if $F \in \mathcal{C}$ and $G \in \mathcal{C}$ imply $H(f, x) := F(f, G(f, x)) \in \mathcal{C}$.
- $\mathcal{C}$ is closed under type-1 composition if $F \in \mathcal{C}$ and $G \in \mathcal{C}$ imply $H(f, x) := F(\lambda y.G(f, y), x) \in \mathcal{C}$.

Why is this hard? Basically, type-0 composition is standard (first-order) function composition, while type-1 composition is closure by oracles. Then, let us consider the class FLOGSPACE (or FL) of the functions computable in logarithmic space. While FL is indeed type-0 closed, we have that $\mathsf{FL}^{\mathsf{FL}} = \mathsf{FP}$ [5]. This means that adding the possibility to query an FL oracle dramatically increases the computational power, thus invalidating the type-1 compositionality of FL.

The Machine Model. We start by defining our variant over oracle Turing machines. We will discuss the non-standard aspects below.

Definition 2. A tape stack oracle Turing machine (SOTM) M consists of:

- a two-way, read-only tape;
- a two-way, read/write working tape;
- a one-way, write-only output tape;
- a pushdown stack of pair of tapes $\langle Q_k, A_k \rangle$ such that for the active tapes $\langle Q_{\text{top}}, A_{\text{top}} \rangle$:
  - Q_{top} is one-way and write-only;
  - A_{top} is one-way and read-only;
  - each time M writes a symbol to Q_{top}, the content of A_{top} is erased;
- a finite state control including four special states `PUSH`, `PULL`, `QUERY`, `ANSWER`.

A few explanations are in order. The behavior of SOTM is the one of usual OTM, *i.e.* when the `QUERY` is reached the content x of the active query tape Q_{top} is read, the state changes to `ANSWER` and the string $f(x)$, where f is the oracle, is immediately written on the answer tape A_{top}. We also require that a sort of "independence" between answers and queries. Once we start to write a query, the answer tape is immediately wiped. The novelty here is the presence of a *stack* of query and answer tapes, inspired by Cook's AuxPDA [2] and Wilson's *oracle stacks* [6]. This stack is managed through the states `PUSH` and `PULL` that create/destroy new pairs of tapes, using a stack-based discipline. This way, only the top most pair can be written/read.

Easy Feasible Functionals. Before defining our new complexity class we need a technical definition that constraints the power of oracles.

Definition 3 (Polynomially-Bounded Oracle)**.** A type-1 functional oracle f is polynomially-bounded if there exists a polynomial P such that for all queries y we have $|f(y)| \leq P(|y|)$.

We are now able to define the class which is the type-2 equivalent of FLOGSPACE.

Definition 4 (Easy Feasible Functionals (EFF))**.** A functional F is in EFF if and only if there exists a SOTM M that computes F such that for each type-0 input x and type-1 polynomially-bounded oracle f[1], the maximum number of cells written on the working tape is $\mathcal{O}(\log |x|)$ and the maximum height of the tape stack is $\mathcal{O}(1)$.

The interesting thing to note here is that, unlike the case of BFF, we do not need second-order polynomials. This is for a very precise reason: while polynomials perfectly compose, this is not the case for logarithms of polynomials. Moreover, please note the restriction to the maximum stack height to a constant.

Theorem 1.

1. EFF *are closed under type-0 composition.*
2. EFF *are closed under type-1 composition.*

Proof ingredients.

1. Here the constant height tape stack plays a crucial role in the simulation.
2. Here we exploit (i) the fact that answer tapes get erased when we start to write a new query and (ii) the polynomial bound on oracles. □

[1]The restriction to polynomially-bounded oracles could seem either very strong or arbitrary. Actually, it this restriction turns out to be equivalent to devising a SOTM that can ask for the content of the oracle query $f(y)$ bit-by-bit, via pointers. This mechanism is standard when dealing with LOGSPACE composition.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[2] Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, 1971.

[3] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996.

[4] Akitoshi Kawamura. *Computational Complexity in Analysis and Geometry*. PhD thesis, University of Toronto, Canada, 2011.

[5] Richard E. Ladner and Nancy A. Lynch. Relativization of questions about log space computability. *Math. Syst. Theory*, 10:19–32, 1976.

[6] Christopher B. Wilson. A measure of relativized space which is faithful with respect to depth. *J. Comput. Syst. Sci.*, 36(3):303–312, 1988.

Counting Complexity of ASP

Max Bannach[1] , **Johannes K. Fichte**[2] , **Johanna Groven**[2] , **Markus Hecher**[3]
[1]European Space Agency, The Netherlands
[2]Linköping University, Sweden
[3]CNRS, CRIL, France
max.bannach@esa.int, firstname.lastname@liu.se, hecher@cril.fr

Abstract

Answer Set Programming (ASP) is a mature and widely used framework for modeling and solving problems in AI, knowledge representation and reasoning, and combinatorial search. Counting answer sets is of growing importance for analyzing search spaces, navigating ASP programs, and enabling probabilistic reasoning. While Truszczyński established a complete hierarchy for the computational complexity of ASP decision and reasoning problems (skeptical and credulous), a corresponding systematic treatment of counting problems has been missing so far. We close this gap by providing an almost complete characterisation of the counting complexity landscape for ASP.

1 Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011; Gebser et al. 2012; Janhunen and Niemelä 2016) is a mature and widely used framework for modeling and solving problems in AI with plenty of application areas, including industrial (Nabeshima et al. 2026; Alviano and Reiners 2025; Altay-Kern et al. 2025) and academic applications (Hahn et al. 2024; Baumeister et al. 2024). The computational complexity of decision and optimization problems in ASP is well understood ranging from polynomial-time solvable (Marek and Truszczyński 1991; Gelfond and Lifschitz 1988) to the second level of the polynomial hierarchy (Eiter and Gottlob 1995) for propositional programs. Truszczyński (2011) established a comprehensive trichotomy for decision and reasoning problems (skeptical and credulous) with detailed characterization of subclasses of programs. In the ASP community, these results are widely known as an equivalent of Schaefer's classes in propositional satisfiability (SAT), constraint satisfaction (CSP) (Schaefer 1978; Chen 2006), or Post's lattice in universal algebra (Post 2016; Lau 2006).

While decisions, reasoning, and optimization in ASP have been well studied, counting the number of solutions is becoming increasingly more important due to numerous applications (Darwiche 2020; Zhang et al. 2024; Zak et al. 2025). In logic programming, counting enables probabilistic reasoning without enumerating (Azzolini and Riguzzi 2025; Bogaerts and Van den Broeck 2015; Fierens et al. 2015), and navigating and understanding large search spaces (Fichte, Gaggl, and Rusovac 2022; Kabir and Meel 2023). Surprisingly, the complexity landscape of ASP counting is quite open. While we have results for Horn (Dowling and Gallier 1984), stratified (Gelfond and Lifschitz 1988), tight (Clark 1977), and disjunctive programs (Fichte et al. 2017), contrary to what one might expect, the existing trichotomy results do not carry over to counting.

In this paper, we address this gap by studying the computational complexity of ASP in the well-known trichotomy framework by Truszczyński. It turns out that ASP counting is much harder to comprehend than propositional satisfiability (#SAT). The minimality of disjunctions in rule heads significantly complicates any complexity analysis. We provide our main results in this abstract. We omit any proofs which can be found in the full conference version.

Table 1 presents our main contributions. In particular, we establish a comprehensive overview for answer set counting in the framework of Truszczyński.

2 Preliminaries

We assume that the reader is familiar with basics in propositional logic, ASP, and computational complexity. Below, we summarize notations.

Computational Complexity. We follow standard terminology in computational complexity (Papadimitriou 1994) and the Polynomial Hierarchy (PH) (Stockmeyer and Meyer 1973; Stockmeyer 1976; Wrathall 1976), and counting complexity (Toda and Watanabe 1992; Hemaspaandra and Vollmer 1995; Durand, Hermann, and Kolaitis 2005). We use of complexity classes preceded with the sharp-dot operator '#·', parsimonious and subtractive reductions. Let M be a deterministic Turing machine (DTM) or non-deterministic Turing Machine (NTM) such that for all inputs x, every computation path on x is accepting or rejecting and is bounded in length by a function $t(x)$. For input x, we define $\mathrm{acc}_M(x), \mathrm{rej}_M(x)$ as the number of *accepting and rejecting computation paths* of x on M respectively. Note that $\mathrm{acc}_M(x) + \mathrm{rej}_M(x)$ is the number of all computation paths of x. We define **FP** as the class of all function problems that are polynomial-time reducible to $\mathrm{acc}_M(x)$ for some polytime DTM M, #P as the class of all counting problems that are polynomial-time reducible to $\mathrm{acc}_M(x)$ for some polytime NTM M. Let **RP** be the class of computation prob-

Class C	Name	Cons	#AS	Result
$\{[1,\infty,0],[0,\infty,\infty]\}$	Horn	P	FP	[A]
$\{[\infty,\infty,0],[0,0,0]\}$	NegIntFree	P	no-FPRAS†	Lem 4.3
$\{[\infty,1,0],[0,1,0]\}$	DHorn	P	open	
$\{[i,0,k],[0,j,k] \mid i+k\leq 2, j+k\leq 1\}$		NP-c	no-FPRAS†	Lem 4.3
$\{[i,j,k] \mid i+j+k\leq 2\}$	Krom	NP-c	no-FPRAS†	Lem 4.3
$\{[1,\infty,\infty],[0,\infty,\infty]\}$	Normal	NP-c	#P-c	Lem 4.1, Obs 5
$\{[\infty,1,\infty],[0,\infty,\infty]\}$	DNormal	NP-c	#P-c	Lem 4.1, Obs 5
$\{[\infty,\infty,0],[0,\infty,0]\}$	NegFree	NP-c	#P-c	Lem 4.1, Obs 5
otherwise	Disj	Σ_2^{P}-c	# · coNP-c	Lem 4.4, [B]
	Strat	P	FP	[C]
	Tight	NP-c	#P-c	[D]

Table 1: C refers to the class of programs or the class that consists of $\Delta_P \leq \Delta(C)$. †: unless RP = NP. [A]: (Dowling and Gallier 1984); [B]: (Fichte et al. 2017); [C]: (Gelfond and Lifschitz 1988); [D]: (Clark 1977). Results for Cons can be found in (Truszczyński 2011) and [C], [D].

lems L such that there exists a poly-time randomized TM M such that for $x \in L$, we have that $\Pr[M \text{ accepts } x] \geq 0.5$ and for $x \notin L$, we have $\Pr[M \text{ accepts } x] = 0$. A fully *randomized approximation scheme (**FPRAS**)* for a function $f : \{0,1\}^* \to \mathbb{R}_{\geq 0}$ is a randomized algorithm $A(x,\varepsilon,\delta)$ such that for every $x \in \{0,1\}^*, \varepsilon, \delta > 0$, $A(x,\varepsilon,\delta)$ outputs y such that $\Pr[(1-\varepsilon)f(x) \leq y \leq (1+\varepsilon)f(x)] \geq 1-\delta$ in time polynomial in $|x|, \varepsilon^{-1}, \log(\delta^{-1})$, i.e., A computes at least an ε approximation of $f(x)$ with probability at least $1-\delta$.

Answer Set Programming (ASP) For a comprehensive introduction, we refer to standard texts (Janhunen and Niemelä 2016; Gebser et al. 2012). We restrict ourselves to propositional programs. Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$, and $a_1, \ldots, a_n$ distinct propositional atoms. We let programs consist of *rules*, in particular, *integrity rules* (constraints). A *disjunctive rule* r is of the form $a_1 \vee \cdots \vee a_l \leftarrow a_{\ell+1}, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n$. By $H_r := \{a_1, \ldots, a_l\}$, $B_r^+ := \{a_{l+1}, \ldots, a_m\}$, and $B_r^- := \{a_{m+1}, \ldots, a_n\}$, we denote the *head*, *positive* or *negative body* atoms of r, respectively. A *program* Π is a set of rules, where $\mathrm{at}(\Pi) := \bigcup_{r\in\Pi}(H_r \cup B_r^+ \cup B_r^-)$ denotes its atoms. We call a rule r with both $H_r \neq \emptyset$ and body $B_r^+ \cup B_r^- \neq \emptyset$ *non-simple*. An interpretation $M \subseteq \mathrm{at}(\Pi)$ *satisfies* a disjunctive rule r if $(H_r \cup B_r^-) \cap M \neq \emptyset$ or $B_r^+ \not\subseteq M$. M is a *model* of Π if M satisfies every rule $r \in \Pi$. The *reduct* r^M of a disjunctive rule r is the singleton $\{H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset\}$, and $\Pi^M := \{r' \mid r' \in r^M, r \in \Pi\}$ is called *GL reduct* of Π with respect to M. Then, M is an *answer set* of Π if M is a model of Π such that no interpretation $N \subsetneq M$ is a model of Π^M (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991). We let the set $\mathrm{AS}(\Pi)$ consist of all answer sets of Π. The problem $\mathrm{Cons}(\Pi)$ asks to decide whether $\mathrm{AS}(\Pi) \neq \emptyset$ and $\#AS(\Pi)$ asks to output $|\mathrm{AS}(\Pi)|$.

ASP Subclasses We say that a rule r is *normal* if $|H_r| \leq 1$, *positive* if $B_r^- = \emptyset$, and an *integrity constraint* if $H_r = \emptyset$. Moreover, a program Π has a certain property if all its rules have the property. The *dependency graph* D_Π of Π is the directed graph on the atoms $\bigcup_{r\in\Pi} H_r \cup B_r^+$, where for every rule $r \in \Pi$, two atoms $a \in B_r^+$ and $b \in H_r$ are joined by an edge (a,b). We say Π is *tight* if D_Π is acyclic (Fages 1994). A program Π is called head-cycle free if its incidence graph does not contain any directed cycles (Ben-Eliyahu and Dechter 1994). By Disj, Normal, DNormal, NegFree, Horn, DHorn, NegIntFree, and Tight, we denote the class of all disjunctive, normal, negation-free, dual-normal, Horn, dual-Horn, or tight programs. Truszczyński defined detailed (arity) restrictions given by specifying upper bounds on the arities of the rules used to analyze the computational complexity and establish a Trichotomy (Truszczyński 2011). Therefore, let $U \in \{0,1,2,\ldots\} \cup \{\infty\}$ where ∞ represents unbounded arity and $\mathcal{A} = \{[k,m,n] \mid k,m,n \in U\}$. For $\alpha = [k,m,n] \in \mathcal{A}$, let $\alpha_1 = k, \alpha_2 = m, \alpha_3 = n$. For a program rule r, let $\alpha_r = [|H_r|, |B_r^+|, |B_r^-|]$ be the arity of r. For two restrictions $\alpha, \beta \in \mathcal{A}$, we have $\alpha \preceq \beta$ if and only if (i) $\alpha_1 = 0$, then $\beta_1 = 0$, and (ii) $\alpha_i \leq \beta_i$, for $i \in \{1,2,3\}$ where $\leq$ corresponds to the normal total on the natural numbers extended by $a \leq \infty$ for all $a \in \mathbb{N}_0 \cup \{\infty\}$. Let $\Delta \subseteq \mathcal{A}$. Then we define $C(\Delta)$ to be the class of all *finite* programs P that satisfy the following condition: for every rule $r \in P$ there is $\alpha \in \Delta$ such that $\alpha_r \preceq \alpha$. E.g., $C(\{[1,\infty,\infty],[0,\infty,\infty]\})$ specifies Normal. The decision complexity of subclasses is well-studied with a full characterization based on the arity characteristics.

Proposition 1 (Truszczyński, 2011)**.** *Let* $\Pi \in C(\Delta)$ *be a program. Then,* $\mathrm{Cons}(\Pi) \in$ P *if* $\Delta \preceq \{[1,\infty,0],[0,\infty,\infty]\}$ *or* $\Delta \preceq \{[\infty,1,0],[0,1,0]\}$ *or* $\Delta \preceq \{[i,j.0] \mid i+j \leq 2\}$*; and otherwise* $\mathrm{Cons}(\Pi) \in$ NP*-c, if* $\Delta \preceq \{[1,\infty,\infty],[0,\infty,\infty]\}$ *or* $\Delta \preceq \{[\infty,1,\infty],[0,\infty,\infty]\}$ *or* $\Delta \preceq \{[\infty,\infty,0],[0,\infty,0]\}$*; and otherwise* $\mathrm{Cons}(\Pi) \in \Sigma_2^{\mathrm{P}}$*-c.*

3 ASP Counting Complexity

While counting complexity results for basic classes of ASP programs are known, there is no systematic approach to characterize the complexity based on restrictions of arities in the head, positive body, and negative body. We follow the structural characterization introduced by (Truszczyński 2011) for decision and reasoning problems and investigate the computational complexity of fragments. We present a simplified

overview in Table 1. Before we present our results in detail, we make the following observation.

Observation 2. *Let $\Pi \in C(\{[x,0,0]\} \cup S)$ for some $x \in \mathbb{N}$ and $S \subseteq \mathcal{A}$ such that Π is head-cycle free. Then, for each $k < x$, there exists a program $\Pi' \in C(\{[k,0,x-k]\} \cup S)$ with $AS(\Pi) = AS(\Pi')$ such that Π' is computable in log-space and linear time from Π for every fixed k, x.*

In more detail, we have the following Theorem.

Theorem 3. *Let $\Pi \in C(\Delta)$ be a program where Δ defines a structural restriction on H_r, B_r^+, and B_r^-.*

1. *$\#AS(\Pi) \in$ P if $\Delta_\Pi \preceq \{[1,\infty,0],[0,\infty,\infty]\}$;*
2. *$\#AS(\Pi) \in$ #P if otherwise*
   - *$\{[1,\infty,0],[0,\infty,\infty]\} \preceq \Delta_\Pi$, or*
   - *$\{[i,j,k] \mid i+j+k \leq 2\} \preceq \Delta_\Pi$,*

   and
   - *$\Delta_\Pi \preceq \{[1,\infty,\infty],[0,\infty,\infty]\}$, or*
   - *$\Delta_\Pi \preceq \{[\infty,1,\infty],[0,\infty,\infty]\}$, or*
   - *$\Delta_\Pi \preceq \{[\infty,\infty,0],[0,\infty,0]\}$;*
3. *$\#AS(\Pi) \in \# \cdot$ coNP, otherwise.*

Before we can establish this theorem, we systematically investigate which cases yield hardness and establish the following lower bounds.

Lemma 4. *Let $\Pi \in C(\Delta)$ be a program. Then, $\#AS(\Pi)$*

1. *is* #P*-hard if Δ is any of*
   - *(a)* $\{[3,0,0],[0,2,0]\}$,
   - *(b)* $\{[2,0,1],[0,2,0]\}$, $\{[1,0,2],[0,2,0]\}$,
   - *(c)* $\{[2,0,0],[0,2,0],[0,0,3]\}^\star$
   - *(d)* $\{[2,1,0],[0,2,0]\}^\star$, $\{[2,0,0],[0,3,0]\}^\star$,
   - *(e)* $\{[2,0,0],[0,3,0]\}^\star$, $\{[1,0,1],[0,3,0]\}^\star$,
   - *(f)* $\{[2,0,0],[0,2,1]\}^\star$,
   - *(g)* $\{[2,0,0],[0,2,0],[0,1,2]\}^\star$,
   - *(h)* $\{[1,2,0],[2,0,0],[0,1,0]\}^\star$,
   - *(i)* $\{[1,2,0],[1,0,1],[0,1,0]\}^\star$,
2. *is* #P*-hard by subtractive reduction if Δ is any of*
   - *(a)* $\{[3,0,0],[1,2,0]\}^\star$,
   - *(b)* $\{[2,0,0],[1,2,0]\}^\star$,
   - *(c)* $\{[1,0,1],[1,2,0]\}^\star$,
   - *(d)* $\{[2,0,0],[0,1,0]\}^\star$,
3. *has no* FPRAS *unless* RP = NP *(and therefore unlikely to be efficiently countable) if Δ is any of*
   - *(a)* $\{[2,0,0],[0,2,0]\}^\star$ *(by reduction from #*2SAT*),*
   - *(b)* $\{[1,0,1],[0,2,0]\}^\star$ *(by reduction from #*2SAT*),*
   - *(c)* $\{[2,0,0]\}^\star$ *(by reduction from #*mon-2SAT*),*
   - *(d)* $\{[1,0,1]\}^\star$ *(by reduction from #*mon-2SAT*),*
4. *$\# \cdot$ coNP-hard if Δ is any of*
   - *(a)* $\{[2,0,0],[1,2,0],[1,0,1]\}^\star$,
   - *(b)* $\{[2,0,0],[1,2,0],[0,0,1]\}^\star$,

Proof Sketch for Case 1. For Case 1, we construct a parsimonious reduction (subtractive 1.j–i) from #3-SAT to *#AS* that transforms a #3-SAT instance into a program Π. In many cases we can apply Obs. 2 . E.g. Case 1b holds by Case 1a and Obs. 2. □

Additionally, we make the following observation by constructing for each case a respective Turing machine that proves the membership of the class in *#P* using a branching approach.

Observation 5. *If Δ is one of:*
$\{[\infty,\infty,0],[0,0,0]\}$,
$\{[i,0,k],[0,j,k] \mid i+k \leq 2, j+k \leq 1\}$,
$\{[i,0,k],[0,j,k] \mid i+j+k \leq 2\}$,
$\{[i,0,k],[0,j,k'] \mid i+k \leq 3, k \leq 2j+k' \leq 1\}$,
$\{[1,\infty,\infty],[0,\infty,\infty]\}$,
$\{[\infty,1,\infty],[0,\infty,\infty]\}$,
$\{[\infty,\infty,0],[0,\infty,0]\}$,
then the $\#AS_C$ problem for $C(\Delta)$ is in #P.

Now, we are in position to establish Theorem 3.

Proof of Theorem 3. Let $\Delta_\Pi \preceq \{[1,\infty,0],[0,\infty,\infty]\}$. Then, $\Pi \in$ Horn and has exactly one answer set exactly if it has a model. Thus, $\#AS(\Pi)$ can be reduced to the satisfiability of Horn-formula in this case, which is known to solvable in polynomial time. Next, assume that $\{[1,\infty,0],[0,\infty,\infty]\} \preceq \Delta_\Pi$, but not $\Delta_\Pi \preceq \{[1,\infty,0],[0,\infty,\infty]\}$. Then, either $\{[1,0,1]\} \preceq \Delta_\Pi$ or $\{[2,0,0]\} \preceq \Delta_\Pi$. In both cases, $\#AS(\Pi)$ is #P-hard by Lemma 4.1e. Now, assume that $\{[i,j,k] \mid i+j+k \leq 2\} \preceq \Delta_\Pi$, but not $\Delta_\Pi \preceq \{[i,j,k] \mid i+j+k \leq 2\}$. Then $S \preceq \Delta_\Pi$ for some $S \in \{[3,0,0],[0,3,0],[0,0,3],[1,2,0],[2,1,0],[1,0,2],[2,0,1],[0,1,2],[0,2,1]\}$. These cases are covered by Lemma 4.1a–1g, 4b, resp. If we have $\Delta_\Pi \preceq \{[1,\infty,\infty],[0,\infty,\infty]\}$, $\Delta_\Pi \preceq \{[\infty,1,\infty],[0,\infty,\infty]\}$, or $\Delta_\Pi \preceq \{[\infty,\infty,0],[0,\infty,0]\}$. Then $\#AS(\Pi) \in$ #P by Observation 5. Following Truszczyński (2011), assume that none of the above inclusions hold. By $\Delta_\Pi \not\preceq \{[1,\infty,\infty],[0,\infty,\infty]\}$, we must have $\{[2,0,0]\} \preceq \Delta_\Pi$. By $\Delta_\Pi \not\preceq \{[\infty,1,\infty],[0,\infty,\infty]\}$, we must have $\{[1,2,0]\} \preceq \Delta_\Pi$. By $\Delta_\Pi \not\preceq \{[\infty,\infty,0],[0,\infty,0]\}$, we must have either $\{[0,0,1]\} \preceq \Delta_\Pi$ or $\{[1,0,1]\} \preceq \Delta_\Pi$. In the former, we obtain $\# \cdot$ coNP hardness by Lemma 4.4b. In the latter, we have $\# \cdot$ coNP hardness by Lemma 4.4a. Lastly, $\#AS(\Pi) \in \# \cdot$ coNP for arbitrary Π (Fichte et al. 2017). □

4 Conclusion and Outlook

We study the counting complexity of answer set programming in the arity framework of Truszczyński (2011) and beyond. There we obtain a comprehensive picture of hardness and membership results and reductions.

While our work is mainly of theoretical nature, we expect a positive impact of our translations on practical tools, possible (counting) proof systems for ASP (Alviano et al. 2019; Fichte, Hecher, and Roland 2022), and applications that require multiple counting (Fichte, Hecher, and Nadeem 2022).

Acknowledgements

The authors are listed in alphabetical order. Research supported by the Austrian Science Fund (FWF) grant J4656; the French Agence nationale de la recherche (ANR) grant ANR-25-CE23-7647; and ELLIIT funded by the Swedish government. Part of the research was carried out while Hecher was a postdoc at MIT and while he was at UC Berkeley while visiting the Simons institute for the theory of computing.

References

Altay-Kern, O.; Dietz, E.; Kuhlmann, I.; and Thimm, M. 2025. Exploring Desirable Configurations in Global Logistics with Heuristic Search in Answer Set Programming. In *Proc. of KR*, 578–587.

Alviano, M., and Reiners, L. A. R. 2025. ASP chef for water waste monitoring. In *Proc. of the 40th Italian Conference on Computational Logic*, CEUR Workshop Proc. CEUR-WS.org.

Alviano, M.; Dodaro, C.; Fichte, J. K.; Hecher, M.; Philipp, T.; and Rath, J. 2019. Inconsistency proofs for ASP: The ASP - DRUPE format. *Theory and Practice of Logic Programming* 19(5–6):891–907.

Azzolini, D., and Riguzzi, F. 2025. Probabilistic answer set programming with discrete and continuous random variables. *Theory and Practice of Logic Programming* (1):1–32.

Baumeister, J.; Herud, K.; Ostrowski, M.; Reutelshöfer, J.; Rühling, N.; Schaub, T.; and Wanko, P. 2024. Towards industrial-scale product configuration. In *Proc. of LPNMR*, 71–84.

Ben-Eliyahu, R., and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.* (1):53–87.

Bogaerts, B., and Van den Broeck, G. 2015. Knowledge compilation of logic programs using approximation fixpoint theory. *Theory and Practice of Logic Programming* (4–5):464–480.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Comm. of the ACM* (12):92–103.

Chen, H. 2006. A rendezvous of logic, complexity, and algebra. *SIGACT News* (4):85–114.

Clark, K. L. 1977. Negation as failure. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, 293–322.

Darwiche, A. 2020. Three modern roles for logic in AI. In *Proc. of PODS*, 229–243.

Dowling, W. F., and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.* (3):267–284.

Durand, A.; Hermann, M.; and Kolaitis, P. G. 2005. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.* (3):496–513.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* (3–4):289–323.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Methods Log. Comput. Sci.* 1(1):51–60.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2017. Answer Set Solving with Bounded Treewidth Revisited. In *Proc. of LPNMR*, 132–145.

Fichte, J. K.; Gaggl, S. A.; and Rusovac, D. 2022. Rushing and strolling among answer sets - navigation made easy. In *Proc. of AAAI*, 5651–5659.

Fichte, J. K.; Hecher, M.; and Nadeem, M. A. 2022. Plausibility reasoning via projected answer set counting - a hybrid approach. In *Proc. of IJCAI*, 2620–2626.

Fichte, J. K.; Hecher, M.; and Roland, V. 2022. Proofs for propositional model counting. In *Proc. of SAT*, volume 236, 30:1–30:24.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D. S.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* (3):358–401.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of LP*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.* (3/4):365–386.

Hahn, S.; Schaub, T.; Martens, C.; Nemes, A.; Otunuya, H.; Romero, J.; and Schellhorn, S. 2024. Reasoning about study regulations in answer set programming. *Theory Pract. Log. Program.* (4):790–804.

Hemaspaandra, L. A., and Vollmer, H. 1995. The satanic notations: Counting classes beyond #P and other definitional adventures. *SIGACT News* 26(1):2–13.

Janhunen, T., and Niemelä, I. 2016. The answer set programming paradigm. *AI Magazine* (3):13–24.

Kabir, M., and Meel, K. S. 2023. A fast and accurate asp counting based network reliability estimator. In *Proc. of LPAR*, 270–287. EasyChair.

Lau, D. 2006. *Function algebras on finite sets: a basic course on many-valued logic and clone theory*. Springer.

Marek, W., and Truszczyński, M. 1991. Autoepistemic logic. *J. of the ACM* (3):588–619.

Nabeshima, H.; Banbara, M.; Schaub, T.; and Soh, T. 2026. The ASP-based nurse scheduling system at the university of yamanashi hospital. *Electronic Proc. in Theoretical Computer Science* 334–348.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

Post, E. L. 2016. The two-valued iterative systems of mathematical logic.(am-5).

Schaefer, T. J. 1978. The complexity of satisfiability problems. In *Proc. of STOC*, 216–226. Association for Computing Machinery.

Stockmeyer, L. J., and Meyer, A. R. 1973. Word problems requiring exponential time. In *STOC'73*, 1–9. Assoc. Comput. Mach.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theo. Comput. Science* 3(1):1–22.

Toda, S., and Watanabe, O. 1992. Polynomial-time 1-turing reductions from #PH to #P. *Theoretical Computer Science* 100(1):205–221.

Truszczyński, M. 2011. Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. *Theory Pract. Log. Program.* (6):881–904.

Wrathall, C. 1976. Complete sets and the polynomial-time hierarchy. *Theo. Comput. Science* 3(1):23–33.

Zak, D.; Mei, J.; Lagniez, J.; and Laarman, A. 2025. Reducing quantum circuit synthesis to #SAT. In *Proc. of CP*, 38:1–38:21.

Zhang, H.; Kung, P.-N.; Yoshida, M.; Van den Broeck, G.; and Peng, N. 2024. Adaptable logical control for large language models. In *Advances in Neural Information Processing Systems*, 115563–115587. Curran Associates, Inc.

Characterizing levels of computational complexity by restrictions on hypothetical reasoning

Marcel Ertel[a,*], Eduardo Skapinakis[a,b]

[a] *Carl Friedrich von Weizsäcker-Center, University Tübingen, Germany*
[b] *Center for Mathematics and Applications (NOVA Math), NOVA School of Science and Technology (NOVA FCT), Portugal*

In [2] (currently under review at *Annals of Pure and Applied Logic*), we used applicative theories (see [3]) to give machine-independent characterizations of classes of computational complexity. Strahm [6, 7] introduced the theory PT, whose class of provably total functions coincides with the class of functions computable in polynomial time. We strengthen PT with forms of *tree induction*. Inspired by a recursion scheme of Oitavem [4, 5], we introduce a new induction axiom, $(\mathsf{TRI}^{\vee}_{\mathsf{W}})$, which directly represents the behavior of a non-deterministic Turing machine, and a new theory, PHT, characterizing the polynomial time hierarchy of functions.

Characterizing the individual levels of this hierarchy is neither possible by restricting the logical complexity of the induction scheme nor by simply restricting the number of allowed applications of induction. The underlying reason is that the combination of its applicative base together with first-order logic allows PHT to reason about functionals of higher type.

By restricting the underlying logic of PHT in such a way that modus ponens cannot be applied to premises of the form $(\forall x)(\phi(x) \to \psi(x)) \to \chi$, which corresponds to restricting reasoning about functionals, we define a new theory, PHT_+, and show that the functions in the levels $\Box^p_{k+1}$ can be characterized as those which are provably total in PHT_+ with at most k *occurrences* of $(\mathsf{TRI}^{\vee}_{\mathsf{W}})$ in a Hilbert-calculus derivation.

Interestingly, a related phenomenon can be observed in Heyting arithmetic (HA). Burr [1] provided a separation of the subsystems $i\Phi_{n+1}$, which characterize of the levels of the hierarchy of provably recursive function of HA (the $\prec \omega_{n+2}$-descent-recursive functions). Unlike the subsystems Π_{n+1} of classical Peano Arithmetic (HA), Burr has shown that the strength of their corresponding intuitionistic theories does not depend solely on the depth of nestings of quantifiers, but requires the iterated left-nesting of implications (which, again, allows reasoning about functionals of higher types). We show that—in certain respects similarly to the case of PHT—it is possible to get a characterization of the levels of the computational content of HA in terms of restrictions and stratifications on the underlying logic, while leaving the induction axioms unrestricted. We discuss the similarities and differences between the cases of PHT and HA.

References

[1] Burr, Wolfgang. Fragments of Heyting Arithmetic. The Journal of Symbolic Logic, 65(3):1223–1240, 2000.

*Corresponding author.
Email address: marcel.ertel@uni-tuebingen.de; marcelertel@gmail.com (Marcel Ertel)

[2] Ertel, Marcel and Skapinakis, Eduardo. Non-deterministic operations in applicative theories. Under review at: *Annals of Pure and Applied Logic*.
[3] Jäger, Gerhard and Kahle, Reinhard and Strahm, Thomas. On Applicative Theories. Logic and Foundations of Mathematics, 83–92, 1999.
[4] Oitavem, Isabel. A recursion-theoretic approach to NP. Annals of Pure and Applied Logic 162(8):661–666, 2011.
[5] Oitavem, Isabel. The polynomial hierarchy of functions and its levels. Theoretical Computer Science, 900: 25–34, 2022.
[6] Strahm, Thomas. Polynomial time operations in explicit mathematics. The Journal of Symbolic Logic, 62(2): 575–594, 1997.
[7] Strahm, Thomas. Theories with self-application and computational complexity. Information and Computation, 185(2): 263–297, 2003.

Notes on Sequential Theories

Juvenal Murwanashyaka

May 11, 2026

The concept of sequential theories was introduced by Pudlák [4] in the study of degrees of local interpretability. Sequential theories are theories with a coding machinery, for all objects in the domain of the theory, sufficient for developing partial satisfaction predicates for formulas of bounded depth-of-quantifier-alternations (see Visser [7]). Formally, a first-order theory T is *sequential* if it directly interprets adjunctive set theory AS; in this talk, direct interpretations are 1-dimensional unrelativized interpretations with absolute equality. A related but strictly weaker notion is the class of *Vaught theories*, that is, theories that directly interpret a weak set theory VS of Vaught [5], a non-finitely axiomatizable fragment of AS.

This talk will be based on two preprints [2], [3]. Paper [2] concerns Robinson arithmetic Q. Visser showed that Robinson's Q and Gregorczyk's concatenation theory TC are not sequential by showing that they are not even poly-pair theories. In [2], I consider the theory $\mathsf{Q} + \Theta$ we obtain by extending Q with the axiom Θ asserting that

$$\pi(x, y) = (x + y)^2 + x$$

is a pairing function. I show that $\mathsf{Q} + \Theta$ is not sequential by showing that it is not even a Vaught theory. The proof builds tree-like nonstandard models and uses Ehrenfeucht–Fraïsse games to rule out a direct interpretation of Vaught's weak set theory.

Paper [3] concerns Friedman's weak set theory WD, whose axioms allow adjunction and deletion of single elements. Friedman [1] used WD to characterize sequential theories by direct interpretability, and WD itself is sequential, but the usual witness uses a parameter. In [3], I answer a question of Visser by proving that WD is not parameter-free sequential. In fact, the stronger theory $\mathsf{WD} + \mathsf{EXT} + \mathsf{BU} + \mathsf{BI}$ has a model $\mathcal{V}^\star$ such that $(\mathcal{V}^\star, a) \cong (\mathcal{V}^\star, b)$ for any two elements $a, b \in \mathcal{V}^\star$. Consequently, no parameter-free definable binary relation can uniformly supply the membership relation of AS in all models. Thus WD is essentially parametrically sequential. BU and BI are the binary union axiom and the binary intersection axiom , respectively.

References

[1] Harvey Friedman, Interpretations, according to Tarski, Nineteenth Annual Tarski Lectures (2007).

[2] Juvenal Murwanashyaka, A non-sequential arithmetical theory with pairing (2025). doi:10.48550/arXiv.2509.15191.

[3] Juvenal Murwanashyaka, Friedman's WD is not parameter-free sequential (2025). doi:10.48550/arXiv.2509.14222.

[4] Pavel Pudlák, Some Prime Elements in the Lattice of Interpretability Types, Transactions of the American Mathematical Society, 280(1), 255–275 (1983).

[5] Robert Lawson Vaught, Axiomatizability by a Schema, The Journal of symbolic logic, 32(4), 473–479 (1967).

[6] Albert Visser, What is sequentiality? in: New Studies in Weak Arithmetics, edited by P. Cégielski, C. Cornaros, and C. Dimitracopoulos, CSLI Lecture Notes Vol. 211, pp. 229–269 (2013).

[7] Albert Visser, The small-is-very-small principle, Mathematical Logic Quarterly, 65, 453-478 (2019).