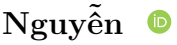
From normal functors to logarithmic space queries



Lê Thành Dũng Nguyễn
LIPN, UMR 7030 CNRS, Université Paris 13, Sorbonne Paris Cité, France
https://nguyentito.eu/
nltd@nguyentito.eu

Pierre Pradic
ENS de Lyon, Université de Lyon, LIP, France
University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Poland
http://perso.ens-lyon.fr/pierre.pradic/
pierre.pradic@ens-lyon.fr

Abstract

We introduce a new approach to implicit complexity in linear logic, inspired by functional database query languages and using recent developments in effective denotational semantics of polymorphism. We give the first sub-polynomial upper bound in a type system with impredicative polymorphism; adding restrictions on quantifiers yields a characterization of logarithmic space, for which extensional completeness is established via descriptive complexity.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory

Keywords and phrases coherence spaces, elementary linear logic, semantic evaluation

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Lê Thành Dũng Nguyễn*: Partially supported by the Elica project (ANR-14-CE25-0005). *Pierre Pradic*: Partially supported by the the RAPIDO project (ANR-14-CE25-0007).

Acknowledgements L. T. D. Nguyễn wishes to thank Damiano Mazza, Thomas Seiller and Kazushige Terui for highly instructive discussions. P. Pradic thanks Alexis Ghyselen for his valuable feedback on a first draft of this paper.

1 Introduction

Machine-free complexity We pursue here a research theme advocated by Leivant [26]: using type systems and the proofs-as-programs correspondence to define functional languages whose expressible functions are exactly those of a given complexity. This usually consists of two independent parts: *soundness* – all those functions admit such complexity bounds – and *extensional completeness* – for every algorithm with this complexity, there is an expressible program computing the same function. This is part of the general area of *implicit computational complexity* (ICC), whose goal is to obtain characterizations of complexity classes by programming languages, without explicit resource bounds on a machine model (other methods in ICC include, for instance, recursive function algebras).

On the other hand, *descriptive complexity* is closer to a declarative programming paradigm: it consists in characterizing complexity classes as sets of *queries* – predicates over finite first-order relational structures – written in some logic. (Such structures often go by the name of *finite models*; see Definition 3.) The field was launched by Fagin's result that NP queries correspond to existential second-order logic [11]. For our purposes, an useful example is Immerman's characterization of *deterministic logarithmic space* (L) (Theorem 13).

This idea of representing inputs as finite first-order structures also appeared in the early history of ICC: at the same FOCS'83 meeting, Gurevich [17] showed that in this setting, a form of primitive recursion captures L, and Leivant [26] deduced from this a characterization

of L in a predicative λ-calculus. But unlike in descriptive complexity, Gurevich considers endofunctions instead of relations and queries.

Queries in the λ-calculus Hillebrand's PhD thesis [18] is a junction point between implicit and descriptive complexity. The idea was to represent finite models inside the simply typed λ-calculus (STλ), using them to represent the inputs to programs. By doing so, Hillebrand et al. managed to characterize P [19], PSPACE [1] and k-EXPTIME/k-EXPSPACE[1] [20] – the extensional completeness for the first two being established through descriptive complexity.

Keeping in mind the connections between finite model theory and relational databases, this can also be seen as using STλ as a functional language for database queries, expressive enough to admit translations from other languages such as Datalog, as is done in [21].

The present paper could then be motivated as looking for a *sub-polynomial*[2] functional query language, filling a gap in the aforementioned work.

Linear logic for ICC Here it is natural to turn to *linear logic*, a constructive logic born from the proofs-as-programs correspondence, in which several characterizations of sub-polynomial complexity classes have already been devised [38, 36, 7, 28, 29]. From its inception, linear logic has indeed had the ambition to "help us improve the efficiency of programs" [13, p. 3], and a landmark result in that direction was characterizing P through Light Linear Logic [16].

In this paper, we will use Elementary Linear Logic (ELL) [16, 8], which was originally introduced to capture the class ELEMENTARY[3]. A recent line of work by Baillot et al. [2, 3, 4] shows that one can define, inside variants of ELL, types of programs which compute smaller complexity classes, such as P. We follow this approach, by introducing a type `Inp` which is essentially an abstract data type[4] for finite models. Our main result is (writing $\mathtt{Bool} = 1 \oplus 1$):

▶ **Theorem 1.** *The class of queries computed by the proofs of* $\mathtt{Inp} \multimap !!\mathtt{Bool}$ *in second-order Elementary Linear Logic* (ELL_2) *is* between L and NL. *Furthermore, a suitable restriction on the existential witnesses in the proof gives an* exact characterization of L.

Here NL stands for *non-deterministic logarithmic space*. Actually, we obtain a better upper bound than NL in the unrestricted case, namely the class L^{UL} which will be defined later. But we believe that this is still not optimal:

▷ Conjecture 1. Even without the restriction, the class of queries obtained is *exactly* L.

Our characterization has a few distinctive features with respect to the previous variants of linear logic capturing logarithmic space [36, 7, 28]: it takes place in a simple pre-existing logical system, which contains only usual logical connectives, and no primitive datatypes[5]; at the price of a more involved encoding of inputs, the `Inp` type. But a main novelty, in our

[1] k-EXPTIME(resp. k-EXPSPACE) is the class of functions which can be computed in time (resp. space) $2 \uparrow^k (p(n))$, where p is a polynomial and n is the size of the input. (We use Knuth's up-arrow notation [24] for iterated exponentials: $2 \uparrow^{k+1} (n) = 2^{2\uparrow^k(n)}$, and $2 \uparrow^0 (n) = n$.)

[2] That is, capturing a complexity class below P. To be fair, Hillebrand's thesis does define a characterization of the sub-polynomial class of *first-order queries* (FO) in STλ, but this class has very little expressivity, and our work captures a class still well above FO.

[3] This is the class of elementary recursive functions, i.e. the union over $k \in \mathbb{N}$ of the classes k-EXPTIME.

[4] This term is the programming language counterpart of existential formulas in logic, cf. infra.

[5] Given the special status granted to unary Church integers by the "skewed iteration" rule in Schöpp's SBAL [36], it is fair to consider them to be primitive datatypes.

opinion, is the unrestricted case: to our knowledge, it is the first[6] sub-polynomial bound in a type system with *impredicative polymorphism*.

This forces our approach to be significantly different to these previous works: they all exploit some form of the Geometry of Interaction (GoI) [14, 9] as a space-efficient evaluator, whereas in our case this does not work[7] because of impredicative quantication. In the predicative case, there is still an obstruction to the GoI: the *additive* connectives of linear logic. Instead, our tool of choice will be *denotational semantics*.

Semantic evaluation and polymorphism This is indeed the sequel to a previous paper [32] which studied the semantics of second-order Multiplicative-Additive Linear Logic (MALL_2) with applications in mind; in particular it proved that Girard's model of MALL_2 in *coherence spaces* [12, 13] is finite and effective. In order to establish our upper complexity bounds, we will compute the denotation of a program applied to its input in the coherence space model.

This *semantic evaluation* technique has been very successful before for establishing complexity bounds in STλ: it is how soundness is established in the aforementioned works of Hillebrand et al., and also underlies Terui's more recent result on the complexity of β-reduction in STλ at fixed order [39]. Beyond STλ, it has been applied to System T and PCF, see [25] and references therein. However, these applications had been confined to monomorphic type systems[8] until the prequel showed:

▶ **Theorem 2** ([32, Thm. III.4]). *The languages decided by proofs of* $!\mathtt{Str} \multimap !!\mathtt{Bool}$ *in* ELL_2*, where* `Str` *is the type of* ELL *Church encodings of strings, are exactly the* regular languages.

An analysis of the proof also suggested that to increase the expressivity[9] while keeping $!!\mathtt{Bool}$ as output, one should replace `Str` by an *existential* input type. Hence the `Inp` type.

To perform semantic evaluation in a polymorphic language, one needs an effective model of polymorphism, and such models are not easy to build. First, one must first restrict to a purely linear language[10] such as MALL_2 to make a non-trivial finitary semantics possible. Even then, obstacles remain: for instance, the prequel [32] proved that no degenerate model of MALL_2 (in which $\otimes$ and $\parr$ are identified) can satisfy a desirable "constancy property", so this excludes the Scott model of linear logic used by [39]. Girard managed to build a semantics for System F [12] which later turned out to be finite and effective for MALL_2 by representing types depending on type parameters as *normal functors*[11]. Although we will not

[6] Excluding the characterization of regular languages in ELL_2, cf. infra, but regular languages do not form a well-behaved complexity class (for instance they are not closed under uniform AC^0 reductions).

[7] We will not enter into details here, but essentially, the GoI works by "following paths" inside a proof, and in our case, the length of these paths would be super-polynomial.

[8] That said, there have been some uses of rather different semantic techniques for implicit complexity in presence of polymorphism, e.g. realizability [6].

[9] This was also a major motivation in the work of Hillebrand et al.: they wanted to overcome limits in STλ such as Statman's classical result that equality cannot be defined on STλ Church integers (see the introduction to [21]). Hillebrand and Kanellakis [20] later proved that the languages decided by STλ predicates over Church-encoded strings are regular (this inspired the analogous result on ELL_2). Such restrictions seem drastic since the β-equivalence problem for STλ is not in ELEMENTARY [37, 27], hinting that its computational power should be much greater. By using finite models as inputs, Hillebrand, Kanellakis and Mairson [21] manage to express all ELEMENTARY queries.

[10] The type $\forall X.\ X \to (X \to X) \to X$ of polymorphic Church integers – more generally, any infinite data type whose destructors are definable – has an inifinite denotation in any semantics of System F.

[11] A remark: the fact that our L^{UL} upper bound involves *unambiguous* nondeterminism, as we shall see, is related to the *stability* of linear maps in coherence spaces; stable maps are the "lower-dimensional analogue" of normal functors, and interestingly, it seems that stability is required for the construction of models of polymorphism based on normal functors.

have to study the properties of normal functors here – the semantic groundwork has been laid in the prequel – we consider that this ingredient is crucial enough to deserve inclusion in the title.

New complexity phenomena in MALL The bottleneck for this L^{UL} bound is the complexity of an iterated composition problem: given a MALL_2 type A and k proofs $f_1, \ldots, f_k$ of $A \vdash A$, compute their composition $f_1 \circ \ldots \circ f_k$. To illustrate the kind of complexity constraint induced by the linearity of the f_i, consider the types $\mathtt{Bool} \otimes \ldots \otimes \mathtt{Bool}$ (n times) and $\mathtt{Bool} \,\&\, \ldots \,\&\, \mathtt{Bool}$ (n times). A non-linear function does not distinguish them, whereas for linear functions:

- an iteration over $\mathtt{Bool} \otimes \ldots \otimes \mathtt{Bool}$ can simulate a Turing machine running in space n (minus $O(1)$ bits for the control state);
- an iteration over $\mathtt{Bool} \,\&\, \ldots \,\&\, \mathtt{Bool}$ can be computed in space $O(\log(nk))$.

This kind of phenomenon surfaced when we tried to obtain bounds on our ELL_2 queries; we are not aware of a previous mention in the literature. Coherence spaces are sensitive to this (e.g. the interpretation of $\otimes$ and $\&$ bit vectors have respective sizes 2^n and $2n$) and thus manage to give a systematic sub-polynomial (but not L) bound on iterations.

For now, we have only managed to find a logarithmic space algorithm for those iterations in very specific cases of A, subsuming the above example. These cases still leave enough room for an extensional completeness result, leading to our exact characterization of L. But even in propositional MALL, the complexity of iterations remains mysterious.

Plan of the paper In Section 2 we introduce the necessary definitions and state the main theorems. The lower bound on expressivity is established using descriptive complexity in Section 3, while our upper bounds are both proved in Section 4 via semantic evaluation.

2 Elementary Linear Logic as a query language

2.1 Linear Logic

In this paper, we assume some familiarity with the basic ideas of the proofs-as-programs paradigm and more specifically of linear logic. The formulas and the sequent calculus of second-order Multiplicative-Additive Linear Logic (MALL_2) are recalled in Appendix A. Recall that MALL_2 forbids using the structural rules of contraction and weakening, enforcing *linearity* whose computational meaning is that data cannot be duplicated or erased.

In order to allow the use of the structural rules in a controlled manner, the grammar of full Linear Logic extends the syntax of MALL_2 with *exponential modalities* $!F$ and $?F$ which allow to tag duplicable assumptions and conclusions. (Second-order) *Elementary Linear Logic* (ELL_2) corresponds to the subsystem whose rules governing the exponential connectives are given in Figure 1; this makes the principles of digging ($!A \multimap !!A$) and dereliction ($!A \multimap A$) invalid in ELL_2 while they are provable in full Linear Logic.

ELL_2 thus satisfies a *stratification* property: the *depth* of a given connective – i.e. the number of !/? modalities it is in the scope of – does not change during cut-elimination (key cut-elimination rules are also recalled in Appendix A). As a consequence, this notion of depth is of the utmost relevance for the computational complexity properties of ELL_2.

LL notations When π and ρ have respective conclusions $\vdash \Gamma, A$ and $\vdash A^{\perp}, \Delta$, we write $\mathbf{cut}(\pi, \rho)$ for the proof of $\vdash \Gamma, \Delta$ consisting of a cut-rule with premises π and ρ. Given a

$$\text{(functorial promotion)}\frac{\vdash \Gamma, A}{\vdash ?\Gamma, !A} \qquad \text{(weakening)}\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \text{(contraction)}\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}$$

Figure 1 Exponential rules for the ELL_2 sequent calculus. In the functorial promotion rule, when $\Gamma = B_1, \dots, B_k$, $?\Gamma$ stands for $?B_1, \dots, ?B_k$.

proof $\pi : A$, $!\pi$ denotes the proof of $!A$ obtained by applying the promotion rule to π. As we formally use one-sided sequents, $A_1, \dots, A_n \vdash B$ is a notation for $\vdash A_1^\perp, \dots, A_n^\perp, B$.

2.2 Finite models

▶ **Definition 3.** *Let Σ be a first-order relational signature, i.e. a list of relation symbols $\{\mathcal{R}_0, \dots, \mathcal{R}_k\}$ with their respective arities $r_0, \dots, r_k$.*

A finite model $\mathfrak{D}$ *over Σ consists of a finite set D and an interpretation $\mathcal{R}_i^{\mathfrak{D}} \subseteq D^{r_i}$ for each relation symbol. It is* totally ordered *when $\mathcal{R}_0 = \leq$, $r_0 = 2$ and $\mathcal{R}_0^{\mathfrak{D}}$ is a total order.*

We write $\mathrm{FinMod}(\Sigma)$ *for the set of totally ordered finite models over Σ.*

As an example, a possible signature for binary strings is $\{\leq, S\}$ with arities 2 and 1. Finite models consist of a totally ordered set $(D, \leq^{\mathfrak{D}})$ with a unary predicate $S^{\mathfrak{D}}$; we interpret $(D, \leq^{\mathfrak{D}})$ as the indices of the string, and $S^{\mathfrak{D}}(d)$ as "the dth bit is set to 1".

▶ Remark 4. The "totally ordered" assumption is common in descriptive complexity (see e.g. Theorem 13) and will be often kept implicit in the paper. Indeed, there are order-independent queries requiring a total order to be expressed.

To use finite models as inputs for ELL_2 programs, we represent the elements of $\mathrm{FinMod}(\Sigma)$ as proofs of an ELL_2 formula $\mathtt{Inp}_\Sigma$.

▶ **Definition 5.** *We define the types with a free variable δ:*

$$\mathtt{List}[\delta] = \forall X.\, !(\delta \multimap X \multimap X) \multimap !(X \multimap X) \qquad \mathtt{C}[\delta] = \delta \multimap \delta \otimes \delta \qquad \mathtt{W}[\delta] = \delta \multimap 1$$

$$\mathtt{Ctx}[\delta] = !\mathtt{List}[\delta] \otimes !!\mathtt{C}[\delta] \otimes !!\mathtt{W}[\delta] \qquad \mathtt{Bool} = 1 \oplus 1 \qquad \mathtt{Rel}_r[\delta] = \delta^r \multimap \mathtt{Bool}$$

Given a signature $\Sigma = \{\leq, \mathcal{R}_1, \dots, \mathcal{R}_k\}$ with arities $r_0 = 2, r_1, \dots, r_k$, we also define:

$$\mathtt{Inp}_\Sigma[\delta] = \mathtt{Ctx}[\delta] \otimes \bigotimes_{0 \leq i \leq k} !!\mathtt{Rel}_{r_i}[\delta] \qquad \mathtt{Inp}_\Sigma = \exists \delta.\, \mathtt{Inp}_\Sigma[\delta]$$

We now define the encoding $\overline{\mathfrak{D}}$ of any totally ordered finite model $\mathfrak{D}$ over Σ as a proof of $\mathtt{Inp}_\Sigma[\mathtt{Fin}(n)]$, where $\mathtt{Fin}(n) = 1 \oplus \dots \oplus 1$ with n summands, n being the domain size.

Let $\mathfrak{D} = (D, \leq^{\mathfrak{D}}, \mathcal{R}_1^{\mathfrak{D}}, \dots, \mathcal{R}_k^{\mathfrak{D}}) \in \mathrm{FinMod}(\Sigma)$ with $\mathrm{Card}(D) = n$. Choose a bijection between D and the n proofs of $\mathtt{Fin}(n)$.

- *We represent D as a Church-encoded list of type $\mathtt{List}[\mathtt{Fin}(n)]$ enumerating the n elements of $\mathtt{Fin}(n)$.*
- *Each relation $\mathcal{R}_i^{\mathfrak{D}}$ can be represented by an element of $\mathtt{Rel}_{r_i}[\mathtt{Fin}(n)]$.*
- *Finally, since $\mathtt{Fin}(n)$ is a positive type, there are canonical elements of type $\mathtt{C}[\mathtt{Fin}(n)]$ and $\mathtt{W}[\mathtt{Fin}(n)]$ implementing the structural rules.*

▶ **Definition 6.** *A proof π of $\mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool}$ defines the query which evaluates to true on $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$ iff the application of π to the encoding $\overline{\mathfrak{D}}$ reduces to $!!\mathtt{true}$ (where $\mathtt{true}$ is the proof of $\mathtt{Bool} = 1 \oplus 1$ proving the left occurrence of 1).*

2.3 Complexity classes and the main theorems

For the rest of the paper, we fix a signature $\Sigma = \{\mathcal{R}_0 = \leq, \mathcal{R}_1, \dots, \mathcal{R}_N\}$ with arities $r_0 = 2, r_1 \dots, r_N$.

As we said in the introduction, we write L (resp. NL) for the class of decision problems solvable in deterministic (resp. non-deterministic) logarithmic space. The *unambiguous* logarithmic space class UL [33] consists of the problems which can be solved by a NL Turing machine whose accepting runs are guaranteed to be *unique*: for each input, if the machine accepts, there is a single sequence of non-deterministic choices leading to the accepting state. (So $\mathsf{UL} \subseteq \mathsf{NL}$.) L^{UL} denotes L with an UL oracle; as usual we use the Ruzzo–Simon–Tompa definition[12] of space-bounded oracle machines [35, §4].

We can now state our result in the unrestricted case.

▶ **Theorem 7.** *The class of queries computed by the proofs of* $\mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool}$ *in* ELL_2 *is between* L *and* L^{UL}.

It is known that $\mathsf{NL}^{\mathsf{NL}} = \mathsf{NL}$ (as noted in [23, Corollary 2], it follows from $\mathsf{NL} = \mathsf{coNL}$), so $\mathsf{L}^{\mathsf{UL}} \subseteq \mathsf{NL}$, hence the statement in the introduction. Furthermore, while $\mathsf{NL} \subset \mathsf{P}$, it is commonly believed that $\mathsf{NL} \neq \mathsf{P}$, so our class of queries is presumably strictly sub-polynomial.

To state the second main theorem, we now introduce a fragment of ELL_2 with an ad-hoc restriction on existential witnesses.

▶ **Definition 8.** *The set of* positive polynomial formulas PP *is the subset of* MALL_2 *formulas generated by the grammar* $P, Q, \dots ::= 0 \mid 1 \mid X \mid P \otimes Q \mid P \oplus Q$.

We define PP3 *to be the set of formulas of the form* $P \otimes (Q \multimap R)$*, where* $P, Q, R \in \mathrm{PP}$.

The logic $\mathrm{ELL}_2^{\mathrm{PP3}}$ *is defined by the same rules as* ELL_2 *except that we exclude the cut rule, and restrict the* $\exists$*-rule as follows: the witness (i.e.* B *in Figure 2) must belong to* PP3.

The "cut-free" part is necessary because *a cut between two* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *proofs does not necessarily normalize into a* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *proof*. However, we do have:

▶ **Proposition 9.** *Let* π *and* ρ *be* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *proofs with respective conclusions* $\vdash \Gamma, A$ *and* $\vdash A^\perp, \Delta$*. If* A *is* quantifier-free*, then* $\mathbf{cut}(\pi, \rho)$ *is in* $\mathrm{ELL}_2^{\mathrm{PP3}}$.

With $\mathrm{ELL}_2^{\mathrm{PP3}}$, we obtain an exact characterization of L:

▶ **Theorem 10.** *The class of queries computed by proofs of* $\mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool}$ *in* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *is* L.

3 The lower bound: encoding logarithmic space queries

In this section, we use descriptive complexity to get the lower bound in both theorems above (so, for the second one, this is an extensional completeness proof).

3.1 Reminder: Immerman's characterization of L

Descriptive complexity considers queries given by formulas in extensions of classical first-order logic. The first-order formulas over Σ are generated by the grammar $\phi, \psi, \dots ::= \mathcal{R}_i(x_1, \dots, x_{r_i}) \mid \neg\phi \mid \phi \vee \psi \mid \exists x.\phi$, where the x_j are variables.

[12] A remark on notation: they would write $\mathsf{L}^{\langle \mathsf{UL} \rangle}$ instead of L^{UL} and use the latter to denote a naive notion of oracle machine. See [35, Example 1] for an example of the subtleties involved: without a careful definition, $\mathsf{NL}^{\mathsf{NL}}$ would include NP.

As usual, the semantics of these formulas is specified by a "satisfaction" relation $\mathfrak{D} \models \phi[\sigma]$ for $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$, defined by induction over ϕ, where σ assigns elements of the domain D of $\mathfrak{D}$ to the free variables of ϕ: e.g. $\mathfrak{D} \models (\exists x.\phi)[\sigma]$ iff $\mathfrak{D} \models \phi[\sigma + (x \mapsto d)]$ for some $d \in D$. Thus, when such a formula ϕ is closed, it defines the query $\mathfrak{D} \mapsto (\mathfrak{D} \models \phi)$.

To express all logarithmic space queries, we need to extend our language of formulas with a *deterministic transitive closure* operator.

▶ **Definition 11.** *The formulas of* first-order logic with deterministic transitive closure (FO+DTC) *are generated by the above grammar extended with a new clause:*

$$\phi, \psi, \ldots ::= \ldots \mid \mathrm{DTC}_{\vec{x},\vec{y}}(\phi) \quad (\vec{x} \text{ and } \vec{y} \text{ are lists of variables of same length})$$

The definition of the satisfaction relation is extended with the following induction case: $\mathfrak{D} \models \mathrm{DTC}_{\vec{x},\vec{y}}(\phi)[\sigma] \iff \sigma(\vec{x})\ R^*\ \sigma(\vec{y})$ *where*

- R^* *is the reflexive transitive closure of the binary relation* $R \subseteq D^k \times D^k$*;*
- D *is the domain of* $\mathfrak{D}$ *and* $\vec{x}, \vec{y}$ *have length* k*;*
- $\vec{a}\ R\ \vec{b} \iff \mathfrak{D} \models \phi_d[\sigma + (\vec{x} \mapsto \vec{a}) + (\vec{y} \mapsto \vec{b})]$[13] *with* ϕ_d *defined as* $\phi \wedge (\forall \vec{z}.\, \phi[\vec{z}/\vec{y}] \Rightarrow \vec{z} = \vec{y})$.

▶ Remark 12. In the above definition, the relation R defined by ϕ_d is *deterministic*, i.e. it is the graph of a partial function $D^k \rightharpoonup D^k$, hence the name. Indeed, it is a "determinization" of the relation defined by ϕ.

▶ **Theorem 13** (Immerman [22])**.** *The* L *queries over totally ordered finite models are exactly those expressible in* FO+DTC*.*

3.2 An encoding of FO+DTC

Thus, it suffices to compile FO+DTC formulas, by induction, to $\mathrm{ELL}_2^{\mathrm{PP3}}$ proofs. For this purpose, it is convenient to interpret formulas with free variables as relation-valued queries:

▶ **Theorem 14.** *Let* $\phi(\vec{x})$ *be an* FO+DTC *formula with* k *free variables. Then there exists an* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *proof* π_ϕ *of* $\mathtt{Inp}[\delta] \vdash \mathtt{!!Rel}_k[\delta]$ *such that, for all* $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$ *with a domain* D *of size* n, $\mathbf{cut}(\overline{\mathfrak{D}}, \pi_\phi[\mathtt{Fin}(n)/\delta])$ *reduces to the encoding of* $\{\vec{a} \in D^k \mid \mathfrak{D} \models \phi[\vec{x} \mapsto \vec{a}]\}$.

▶ **Corollary 15** (Lower bound for Theorem 7 and Theorem 10)**.** *All* FO+DTC *queries – and therefore all* L *queries – over* $\mathrm{FinMod}(\Sigma)$ *can be computed by* $\mathrm{ELL}_2^{\mathrm{PP3}}$ *proofs of* $\mathtt{Inp}_\Sigma \multimap \mathtt{Bool}$.

Proof. Note that $\mathtt{Rel}_0[\delta] \cong \mathtt{Bool}$, and apply a $\parr$-rule and a $\forall$-rule to the $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof given by the previous theorem. ◀

The detailed proof of Theorem 14 is given in Appendix B. As stated before, it works by induction on the FO+DTC formula, the bulk of the work for the induction being the case $\phi = \mathrm{DTC}_{\vec{x},\vec{y}}(\psi)$. The remainder of the section gives a rough summary of the ideas involved.

Let $R \subseteq D^k \times D^k$, and define $\psi_R : Q \mapsto \{(x, z) \mid x = z \vee (\exists y : x\, R\, y \wedge y\, Q\, z)\}$. Then ψ_R is a monotone function over $\mathcal{P}(D^k \times D^k)$, a lattice of height $n^{2k} + 1$ ($n = \mathrm{Card}(D)$). Its least fixpoint $\psi_R^{n^{2k}+1}(\emptyset)$ is exactly the reflexive transitive closure of R. To compute $\psi_R^{n^{2k}+1}$, we use an iterator of type $\mathtt{Nat} = \forall X.\, !(X \multimap X) \multimap !(X \multimap X)$ derived from the $\mathtt{List}[\delta]$.

But this only allows us to iterate *linear* functions. This is where we use the assumption that R is *deterministic*: if $f_R : D^k \rightharpoonup D^k$ is the partial function associated to R, then

[13] The new assignments for $\vec{x}$ and $\vec{y}$ override the pre-existing ones in σ.

$\psi_R(Q) = \{(x,z) \mid x = z \vee (f_R(x) \text{ defined} \wedge f_R(x)\, Q\, z)\}$. In this reformulation, the existential quantifier, which was a source of non-linearity, has disappeared: now, for each (x,z), the evaluation of $(x,z) \in \psi_R(Q)$ uses Q at most once, on $(f_R(x), z)$. In the end, we manage to write a function of type $\mathtt{Rel}_{2k}[\delta] \multimap \mathtt{Rel}_{2k}[\delta]$ representing ψ_R, which we feed to the `Nat`.

A not-quite-trivial step is to define a proof of $\mathtt{Ctx}[\delta], !!\mathtt{Rel}_{2k}[\delta] \vdash !!(\delta^k \multimap 1 \oplus \delta^k)$ sending a relation ϕ to the partial function associated to its determinization ϕ_d. To do so, at one point, we need to instantiate the input $\mathtt{List}[\delta]$ at the type $\delta^{k-1} \otimes (\delta^k \multimap 1 \oplus \delta^k \oplus 1)$; this is our most complicated existential witness, and it is in PP3. We refer the reader to Appendix B again for details.

4 The upper bounds: semantic evaluation

We now give space-efficient algorithms for queries defined by proofs of $\mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool}$ in ELL_2 (resp. $\mathrm{ELL}_2^{\mathrm{PP3}}$). First, we analyse the shape of such a proof, to obtain alternative definitions of the same predicates involving only MALL_2 types and proofs. This puts us in a position to evaluate our queries in a finite, effective semantics of MALL_2: the model of coherence spaces and normal functors which we recall next. Then, we quickly derive the unrestricted L^{UL} bound for Theorem 7, and finally prove L soundness for Theorem 10 thanks to a tricky combinatorial algorithm on coherence spaces.

4.1 Syntactic analysis

Purely syntactic arguments suffice to show that our ELL_2 queries can be captured by a kind of function algebra, defined below. Though it bears some similarities with Gurevich's characterization of L [17] by primitive recursion on finite models, a major difference is that our functions may take arguments which are not just domain elements (that can be coded on $O(\log n)$ bits) but also higher-order data of size $\mathrm{poly}(n)$, e.g. relations. Indeed, linearity serves mainly to tame the complexity in presence of higher order, while it is mostly meaningless on first-order data.

▶ **Definition 16.** *We define inductively, simultaneously for all $(k+1)$-tuples $(A_1, \dots, A_k, B)$ of MALL_2 types with at most one free type variable δ, the classes of functions $\mathcal{C}(A_1, \dots, A_k; B)$ taking as input:*

- *a closed MALL_2 type T (i.e. without free variables)*
- *a list $L = [\tau_1, \dots, \tau_n]$ of proofs of T*
- *a k-tuple of proofs $(\rho_1, \dots, \rho_k)$ with $\rho_i : A_i[T/\delta]$*

and returning a proof of $B[T/\delta]$ as follows:

- *if π is a proof of $A_1, \dots, A_k \vdash B$, then*
  $[(T; L; \rho_1, \dots, \rho_k) \mapsto \mathbf{cut}(\rho_1, \dots \mathbf{cut}(\rho_k, \pi[T/\delta]) \dots)] \in \mathcal{C}(A_1, \dots, A_k; B)$
- (projection) $\Pi_i^k = [(T; L; \rho_1, \dots, \rho_k) \mapsto \rho_i] \in \mathcal{C}(A_1, \dots, A_k; A_i)$
- (composition) *if $f_i \in \mathcal{C}(A_1, \dots, A_k; B_i)$ for $i \in \{1, \dots, l\}$ and $g \in \mathcal{C}(B_1, \dots, B_l; C)$, then*
  $[(T; L; \vec{\rho}) \mapsto g(T; L; f_1(T; L; \vec{\rho}), \dots, f_l(T; L; \vec{\rho}))] \in \mathcal{C}(A_1, \dots, A_k; C)$
- (iteration) *if $f \in \mathcal{C}(A_1, \dots, A_k; \delta \multimap B \multimap B)$, then*
  $[(T; L = [\tau_1, \dots, \tau_n]; \vec{\rho}) \mapsto f(T; L; \vec{\rho})\langle\tau_1\rangle \circ \dots \circ f(T; L; \vec{\rho})\langle\tau_n\rangle] \in \mathcal{C}(A_1, \dots, A_k; B \multimap B)$
  where
  - *$\pi\langle\tau\rangle$ is the partial application of $\pi : T \multimap B[T/\delta] \multimap B[T/\delta]$ to $\tau : T$, to produce a proof of $B[T/\delta] \multimap B[T/\delta]$;*
  - *$\circ$ is the composition of proofs of $B[T/\delta] \multimap B[T/\delta]$ seen as endomorphisms of $B[T/\delta]$.*

▶ **Proposition 17.** *Let π be an* ELL_2 *proof of* $\forall\delta.((!\mathtt{List}[\delta] \otimes !!A_1 \otimes \ldots \otimes !!A_k) \multimap !!B)$*. Then there exists a function* $f \in \mathcal{C}(A_1, \ldots, A_k; B)$ *such that for all* $\rho_i : A_i[T/\delta]$ *(*$i \in \{1, \ldots, m\}$*) and* $\tau_1, \ldots, \tau_n : T$*,* $\mathbf{cut}(![\tau_1, \ldots, \tau_n] \otimes !!\rho_1 \otimes \ldots \otimes !!\rho_k, \pi) = !!f(T; [\tau_1, \ldots, \tau_n]; \rho_1, \ldots, \rho_k)$ *(where the* $[\tau_1, \ldots, \tau_n]$ *on the left is a Church-encoded list in* ELL_2 *of type* $\mathtt{List}[T]$*).*

Moreover, if π is in $\mathrm{ELL}_2^{\mathrm{PP3}}$*, then there is an inductive derivation for f in which all instances of the iteration scheme use a type of accumulators in* PP3*: that is, they are applied to functions in* $\mathcal{C}(\ldots; \delta \multimap P \multimap P)$ *with* $P \in \mathrm{PP3}$*.*

Though the proof of this proposition presents no conceptual difficulty, it is cumbersome and so is relegated to Appendix C. Importantly, it is thanks to the stratification property of ELL_2 that the types involved in the function algebra can be taken in MALL_2: the argument uses the "collapse at depth 2" operation introduced in the prequel to prove [32, Lemma III.6].

▶ Remark 18. The converse also holds: one can map functions in our algebra to ELL_2 proofs.

This can now be specialized to the case $\pi : \mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool}$; indeed,

$$\mathtt{Inp}_\Sigma \multimap !!\mathtt{Bool} \cong \forall\delta.\ !\mathtt{List}[\delta] \otimes !!\mathtt{C}[\delta] \otimes !!\mathtt{W}[\delta] \otimes \bigotimes_{0 \le i \le N} !!\mathtt{Rel}_{r_i}[\delta] \multimap !!\mathtt{Bool}$$

Our ELL_2-definable (resp. $\mathrm{ELL}_2^{\mathrm{PP3}}$-definable) queries can therefore be specified, equivalently, by functions in $\mathcal{C}(\mathtt{C}[\delta], \mathtt{W}[\delta], \mathtt{Rel}_{r_0}[\delta], \ldots, \mathtt{Rel}_{r_N}[\delta]; \mathtt{Bool})$. The next step is to evaluate these functions in the coherence space model.

4.2 The finite semantics of second-order MALL in coherence spaces

We recall key facts about the denotational model of MALL_2 in which we will carry out our semantic evaluation. A comprehensive introduction to this model for propositional MALL may be found in [15], and the extension to MALL_2 is taken from the prequel [32, Section IV].

In this semantics, a formula/type is interpreted as a *coherence space*: an undirected reflexive graph, i.e. a pair $X = (|X|, \frown_X)$ of a set $|X|$ – customarily called the *web* of X – and a symmetric and reflexive relation $\frown_X \subseteq |X| \times |X|$ – its *coherence relation*. Elements $x, y \in |X|$ are called *coherent* when $x \frown_X y$. A *clique* is a subset of pairwise coherent elements of $|X|$; we write $c \sqsubset |X|$ when c is a clique of X. The denotation of a closed type A is a coherence space, and a proof/program $\pi : A$ is interpreted as a clique $[\![\pi]\!] \sqsubset [\![A]\!]$.

$[\![A]\!]$ is defined by induction on A, the connectives $\otimes, \parr, \&, \oplus, (-)^\perp$ being mapped to operations on coherence spaces. The base case depends on an assignment of type variables. So, if A has n type variables, $[\![A]\!]$ is actually a map from n-tuples of coherence spaces to coherence spaces. Similarly, $[\![\pi]\!]$ also depends on such an assignment, and one should write $[\![\pi]\!](X_1, \ldots, X_n) \sqsubset [\![A]\!](X_1, \ldots, X_n)$. To extend the semantics to MALL_2, we interpret quantifiers as sending such "$(n+1)$-parameter spaces" to "n-parameter spaces". The following proposition sums up the properties that will be necessary for our purposes.

▶ **Proposition 19.** *Let A be a* MALL_2 *type with a single free type variable.*

- $[\![A]\!](X)$ *is* finite*, with size polynomial in the size of X when A is fixed* [32, Theorem IV.9]
- $[\![\pi]\!](X)$ *can be computed in logarithmic space when $\pi : A$ is fixed* [32, Proposition IV.19]

Finally, we need to recall the semantic counterpart of cut-elimination, that is, composition of morphisms. A first remark is that $|X \multimap Y| = |X| \times |Y|$. So a clique $c \sqsubset X \multimap Y$ can in fact be seen as a *binary relation* $c \subseteq |X| \times |Y|$. The composition of c with some $c' \sqsubset Y \multimap Z$, seen as morphisms of coherence spaces, is then none other than their *relational composition*. Additionally, the coherence relation ensures the well-known fact that:

▶ **Proposition 20.** *Let $c \sqsubset X \multimap Y$, $c' \sqsubset Y \multimap Z$, $x \in |X|$ and $z \in |Z|$. Then there exists* at most one *$y \in |Y|$ such that $(x, y) \in c$ and $(y, z) \in c'$.*

4.3 The unrestricted case: an unambiguous logarithmic space bound

In this subsection and in the next one, we abbreviate for convenience $[\![\mathtt{Fin}(n)]\!]$, i.e. the n-vertex coherence space with no edges, as $\mathtt{Fin}(n)$. So, if A is a MALL_2 type with a single variable δ, then $[\![A[\mathtt{Fin}(n)/\delta]]\!] = [\![A]\!](\mathtt{Fin}(n))$. Our main theorem here is:

▶ **Theorem 21.** *Let $f \in \mathcal{C}(A_1, \ldots, A_k; B)$. Then $[\![f(T; L; \rho_1, \ldots, \rho_k)]\!]$ is determined by $[\![T]\!]$, $[\![L]\!] = [[\![\tau_1]\!], \ldots, [\![\tau_n]\!]]$ (where $L = [\tau_1, \ldots, \tau_n]$) and $[\![\rho_1]\!], \ldots, [\![\rho_k]\!]$. Furthermore, when f is fixed, $[\![f(T; L; \rho_1, \ldots, \rho_k)]\!]$ can be computed from these denotations in L^{UL}.*

Proof. By structural induction on Definition 16; the first part is an immediate consequence of the functoriality/compositionality of $[\![-]\!]$, so we focus on the complexity. We take care of the base case, where the function comes from a proof $\pi : (A_1, \ldots, A_k \vdash B)$, with Proposition 19 and the fact that relational composition is in L. For the composition scheme, we use the closure of[14] L^{UL} under composition. The iteration scheme is handled by Lemma 22 below. ◀

▶ **Lemma 22.** *Let A be a MALL_2 type with a single type variable. Given $n, k \in \mathbb{N}$, $f_1, \ldots, f_k \sqsubset [\![A \multimap A]\!](\mathtt{Fin}(n))$ and $(u, v) \in |[\![A]\!](\mathtt{Fin}(n))|^2$, whether $(u, v) \in (f_k \circ \ldots \circ f_1)$ can be decided in UL (in the size of the input, which is $\mathrm{poly}(n, k)$).*

Proof. Thanks to Proposition 20, if $v \in (f_k \circ \ldots \circ f_1)(\{u\})$ then there is a *unique* sequence $u_0 = u, u_1, \ldots u_k = v$ such that $u_{i+1} \in f(\{u_i\})$. We successively guess the u_i; at each point, we need only store (u_i, u_{i+1}) to check its presence in f_i. This can be done by a UL Turing machine because each u_i can be stored in space $O(\log n)$: indeed, $|[\![A]\!](\mathtt{Fin}(n))|$ has cardinality polynomial in n (Proposition 19) and there is a natural representation of its points of using $O(1)$ variables in $|\mathtt{Fin}(n)| = \{1, \ldots, n\}$, see [32, Section IV.D]. (Notice that we do not even make use of the coherence relation of $[\![A]\!](\mathtt{Fin}(n))$; its mere existence ensures that the naive NL algorithm is actually UL.) ◀

The upper bound of Theorem 7 follows immediately from Theorem 21 together with:

▶ **Lemma 23.** *Let $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$. Its ELL_2 encoding $\overline{\mathfrak{D}} : \mathtt{Inp}_\Sigma[\mathtt{Fin}(n)]$ (n is the domain size of $\mathfrak{D}$) contains MALL_2 proofs of $\mathtt{C}[\mathtt{Fin}(n)]$, $\mathtt{W}[\mathtt{Fin}(n)]$ and $\mathtt{Rel}_{r_i}[\mathtt{Fin}(n)]$ ($i \in \{1, \ldots, N\}$).*

The denotations of these proofs in the coherence space model can all be computed in L.

4.4 Iterations in deterministic log space for low-complexity types

As can be seen in the proof of Theorem 21, the single crucial point where the complexity of evaluating a query does not seem to fall squarely in L is Lemma 22. By putting the complexity of this iterated composition problem in L when $A \in \mathrm{PP3}$, we will get the L soundness result for Theorem 10.

A first remark is that for $A \in \mathrm{PP3}$, $A[\mathtt{Fin}(n)/\delta] \cong \mathtt{Fin}(P(n)) \otimes (\mathtt{Fin}(Q(n)) \multimap \mathtt{Fin}(R(n)))$ where P, Q, R are polynomials with integer coefficients. The goal becomes to show:

▶ **Theorem 24.** *Let $A \cong \mathtt{Fin}(m) \otimes (\mathtt{Fin}(n) \multimap \mathtt{Fin}(p))$ for some $m, n, p \in \mathbb{N}$. Given $f_1, \ldots, f_k \sqsubset [\![A \multimap A]\!]$ and $(u, v) \in |[\![A]\!]|^2$, whether $(u, v) \in (f_k \circ \ldots \circ f_1)$ can be decided in L.*

[14] Strictly speaking, L^{UL} denotes a class of decision problems, and it is the associated class of function problems $\mathsf{FL}^{\mathsf{UL}}$ which is closed under composition (the usual proof for FL relativizes).

At this point, the proofs start to involve tricky combinatorics on coherence spaces, so this final section of the paper is written for readers familiar with the coherence space model of MALL (but not necessarily its extension to MALL_2). For instance we will often identify cliques $f \sqsubset A \multimap B$ with linear maps from the cliques of A to the cliques of B.

We start with a lemma solving the case $m = 1$, generalizing the example given at the end of the introduction.

▶ **Lemma 25.** *Let* $A = \mathtt{Fin}(n) \multimap \mathtt{Fin}(p)$, $f_1, \dots, f_k \sqsubset A \multimap A$, $\nu, \nu' \in |\mathtt{Fin}(n)|$ *and* $\pi \in |\mathtt{Fin}(p)|$. *There exists* at most one π' *such that* $(\nu', \pi') \in (f_k \circ \ldots \circ f_1)(\{(\nu, \pi)\})$.

Furthermore, there is a logarithmic space algorithm taking $n, p, f_1, \dots, f_k, \nu, \nu', \pi$ *as inputs which decides whether* π' *exists and, if so, finds it.*

Proof. Consider the adjoint maps $f_i^\perp \sqsubset (\mathtt{Fin}(n) \otimes \mathtt{Fin}(p)^\perp \multimap \mathtt{Fin}(n) \otimes \mathtt{Fin}(p)^\perp)$. The graph $\mathtt{Fin}(n) \otimes \mathtt{Fin}(p)^\perp$ has n connected components, which are all cliques (of size p). These $f_i^\perp$ send cliques to (possibly empty) cliques, so for $j \in |\mathtt{Fin}(n)|$, $f_i^\perp(\{j\} \times |\mathtt{Fin}(p)|)$ is either (1) empty or (2) included in some $\{l\} \times |\mathtt{Fin}(p)|$, for l uniquely determined by j. This defines partial maps $\widehat{f_i^\perp} : |\mathtt{Fin}(n)| \rightharpoonup |\mathtt{Fin}(n)|$: in case (1) $\widehat{f_i^\perp}(j)$ is undefined, in case (2) $\widehat{f_i^\perp}(j) = l$.

This allows us to perform a backwards iteration: we define $\nu_k = \nu'$ and, for $i = k, \dots, 1$, $\nu_{i-1} = \widehat{f_i^\perp}(\nu_i)$; ν_0 can be computed in logarithmic space. If ν_0 is undefined or $\nu_0 \neq \nu$, then π' does not exist: we return false.

Otherwise, let us restrict each i-th intermediate $\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)$ to the connected component corresponding to ν_i, and take the corresponding sub-cliques: for $i = 1, \dots, k$, $f_i' = f_i \cap ((\{\nu_{i-1}\} \times |\mathtt{Fin}(p)|) \times (\{\nu_i\} \times |\mathtt{Fin}(p)|))$. Then either $(f_k' \circ \ldots \circ f_1')(\{\pi\})$ is empty, and π' does not exist; or it contains a single element, which is then π'.

Each ν_i is computable in logarithmic space, so $(f_1', \dots, f_k')$ also is; additionally, the computation of $(f_k' \circ \ldots \circ f_1')(\{\pi\})$ from $(f_1', \dots, f_k')$ and π only needs to store a single point of $\mathtt{Fin}(p)$ in working memory, because the cliques of the latter are subsingletons. Since L is closed under L-reductions, we are done. (Making the interactive composition explicit results in a quadratic time algorithm.) ◀

We would like to L-reduce the problem to the case $m = 1$, by determining the projection to $|\mathtt{Fin}(m)|$ of the unique "path" of $k + 1$ points corresponding to a point of the clique $f_k \circ \ldots \circ f_1$. This would involve an iteration analogous to the previous proof, but forwards instead of backwards.

But the image $f_i(\{j\} \times |\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)|)$ is *not necessarily connected*, because $\{j\} \times |\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)|$ is not a clique (though $\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)$ is a connected graph, it is not complete). So one cannot guarantee that this image is included in some $\{l\} \times |\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)|$. An explicit counter-example is the interpretation of the term $\lambda(x \otimes g).\,((g\,x) \otimes \ldots)$ when $m = n$: in some sense, the first component of the output depends on *both* x *and* g *being known*, not only x. However, knowing x is enough to determine what argument will be fed to g (x itself, in this example). The intuitive idea is to propagate this backwards.

The following lemma ensures that we can always either carry on with the forwards iteration or start the backwards propagation (Π_1 (resp. Π_2) is the projection on the first (resp. second) component):

▶ **Lemma 26.** *Let* $c \sqsubset A \parr B$ *be a non-empty clique. Then* $\Pi_1(c)$ *is included in a connected component of* A, *or (non-exclusively)* $\Pi_2(c)$ *is included in a connected component of* B.

Proof sketch. If $\Pi_1(u)$ and $\Pi_1(v)$ are in different connected components for $u, v \in c$, then $\Pi_2(u)$ and $\Pi_2(v)$ are coherent or equal, and all other $\Pi_2(w)$ are coherent or equal to at least one of them: $\Pi_2(c)$ is connected with diameter ≤ 3. ◀

Proof of Theorem 24. We write $A = \mathtt{Fin}(m) \otimes B$ and $B = \mathtt{Fin}(n) \multimap \mathtt{Fin}(p)$. If $n = 0$, $A \cong 0$ and the problem is trivial and if $n = 1$, $A \cong \mathtt{Fin}(m) \otimes \mathtt{Fin}(p)$, so a simple forward propagation solves the problem. From now on, we thus assume that $n > 1$, which makes B connected. Let $f_1, \dots, f_k \sqsubset A \multimap A$, $(\mu, (\nu, \pi)) \in |A|$ and $(\mu', (\nu', \pi')) \in |A|$. The goal is to decide, in logarithmic space, whether $(\mu', (\nu', \pi')) \in (f_k \circ \dots \circ f_1)(\{(\mu, (\nu, \pi))\})$.

Let $\mu_0 = \mu$. If the clique $f_1(\{(\mu, (\nu, \pi))\})$ is empty, then the answer is negative; else, let $\{\mu_1\} \times |B|$ be the connected component containing it. For $1 \leq i < k$, assuming that μ_i is defined, then $f_{i+1}(\{\mu_i\} \times |B|)$ is either:

- empty, and the answer is negative;
- non-empty and contained in some $\{\mu_{i+1}\} \times |B|$ – this defines $\mu_{i+1} \in |\mathtt{Fin}(m)|$ uniquely;
- non-empty and disconnected.

Let $f_i^\ddagger = f_i \cap ((\{\mu_{i-1}\} \times |B|) \times (\{\mu_i\} \times |B|))$ for all $i \geq 1$ for which μ_i is defined. If the iteration reaches $i = k$, this means that $(f_1^\ddagger, \dots, f_k^\ddagger)$ can be computed in logarithmic space, and as in Lemma 25 we can use this to L-reduce the problem to the case $m = 1$, so we are done. If it aborts because of emptiness, then the algorithm can immediately return false.

The remaining case is the last item above. Suppose that μ_{i+1} is undefined because of disconnectedness. Let $f_{i+1}^\dagger = f_{i+1} \cap ((\{\mu_i\} \times |B|) \times |A|)$; it can be seen as a clique $f_{i+1}^\dagger \sqsubset B \multimap A = B^\perp \parr A$, with $B^\perp = \mathtt{Fin}(n) \otimes \mathtt{Fin}(p)^\perp$. The assumption that $\Pi_2(f_{i+1}^\dagger) = f_{i+1}(\{\mu_i\} \times |B|)$ is non-empty and disconnected entails, by Lemma 26, that $\Pi_1(f_{i+1}^\dagger)$ is connected. In other words $\Pi_1(f_{i+1}^\dagger) \subseteq \{\nu''\} \times |\mathtt{Fin}(p)|$ for some ν''.

Let us apply the algorithm of Lemma 25 to the inputs $n, p, f_1^\ddagger, \dots, f_i^\ddagger, \nu, \nu'', \pi$. This can be done in logarithmic space, and the subroutine either raises a failure or gives us some $\pi'' \in |\mathtt{Fin}(p)|$. In the former case, we can return false; in the latter, we know that $(f_k \circ \dots \circ f_1)(\{(\mu, (\nu, \pi))\}) = (f_k \circ \dots \circ f_{i+1})(\{(\mu_i, (\nu'', \pi''))\})$. So all we have to do is to tail-recurse on a suffix of the original input; to implement this in L, it suffices to keep a counter indicating what the current suffix is. This is a strict suffix, because μ_1 is always defined by construction (see above); therefore, our algorithm terminates, while maintaining a logarithmic working space. ◀

▶ Remark 27. $\mathtt{Fin}(m) \otimes (\mathtt{Fin}(n) \multimap \mathtt{Fin}(p)) \cong \bigoplus_{i=1}^m \&_{j=1}^n \bigoplus_{k=1}^p 1$, and such a bicartesian MALL formula can be seen as a game where Player and Opponents alternate choices of branches. Linear implication consists in playing two games in parallel. Morally, Lemma 26 says: if it is your turn to play on both boards, then you must make a choice; and our L algorithm is mostly about scheduling a set of strategies interacting together.

5 Perspective: unrestricted L upper bound through game semantics?

In the extensional completeness proof, strikingly, the *determinism* of a relation corresponds exactly to the *linearity* of its pre-composition operator. This is one reason for which we believe that our class of queries in ELL_2 is exactly L (Conjecture 1) – or at least, that it is strictly containted in NL which corresponds to first-order logic with general transitive closure [22]. Thus, our L^{UL} bound is likely not optimal: it is widely believed that $\mathsf{UL} = \mathsf{NL}$ [34, 33].

To bring down the complexity of the bottleneck – namely the iterated composition – from UL to L, bridging the intuitions of Remark 27 with a proper game semantics of full MALL_2 might be key. In this direction, it is known that the points of the web of a (hyper)coherence space can be seen as external positions of a game [10, 5, 30, 31]. With this point of view, the uniqueness of the intermediate points in the iteration of Lemma 22 reflects the determinism of an underlying interaction which reaches those final positions.

References

1. Serge Abiteboul and Gerd Hillebrand. Space usage in functional query languages. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Georg Gottlob, and Moshe Y. Vardi, editors, *Database Theory — ICDT '95*, volume 893, pages 439–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995. doi:10.1007/3-540-58907-4_33.
2. Patrick Baillot. On the expressivity of elementary linear logic: Characterizing Ptime and an exponential time hierarchy. *Information and Computation*, 241:3–31, April 2015.
3. Patrick Baillot, Erika De Benedetti, and Simona Ronchi Della Rocca. Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. *Information and Computation*, 261:55–77, August 2018. doi:10.1016/j.ic.2018.05.005.
4. Patrick Baillot and Alexis Ghyselen. Combining Linear Logic and Size Types for Implicit Complexity. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2018.9.
5. Pierre Boudes. Projecting Games on Hypercoherences. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, volume 3142, pages 257–268. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-27836-8_24.
6. Ugo Dal Lago and Martin Hofmann. Realizability models and implicit complexity. *Theoretical Computer Science*, 412(20):2029–2047, April 2011. doi:10.1016/j.tcs.2010.12.025.
7. Ugo Dal Lago and Ulrich Schöpp. Computation by interaction for space-bounded functional programming. *Information and Computation*, 248:150–194, June 2016. doi:10.1016/j.ic.2015.04.006.
8. Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, May 2003. doi:10.1016/S0890-5401(03)00010-5.
9. Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1):79–97, September 1999. doi:10.1016/S0304-3975(99)00049-3.
10. Thomas Ehrhard. Parallel and serial hypercoherences. *Theoretical Computer Science*, 247(1):39–81, September 2000. doi:10.1016/S0304-3975(00)00173-0.
11. Ronald Fagin. *Contributions to the model theory of finite structures*. PhD thesis, University of California, Berkeley, 1973.
12. Jean-Yves Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, January 1986. doi:10.1016/0304-3975(86)90044-7.
13. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987. doi:10.1016/0304-3975(87)90045-4.
14. Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Studies in Logic and the Foundations of Mathematics*, volume 127 of *Logic Colloquium '88*, pages 221–260. Elsevier, January 1989.
15. Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*. Cambridge University Press, 1995.
16. Jean-Yves Girard. Light Linear Logic. *Information and Computation*, 143(2):175–204, June 1998. doi:10.1006/inco.1998.2700.
17. Yuri Gurevich. Algebras of feasible functions. In *24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 210–214, Tucson, AZ, USA, November 1983. doi:10.1109/SFCS.1983.5.
18. Gerd G. Hillebrand. *Finite Model Theory in the Simply Typed Lambda Calculus*. PhD thesis, Brown University, Providence, RI, USA, 1994.

19 Gerd G. Hillebrand and Paris C. Kanellakis. Functional Database Query Languages As Typed Lambda Calculi of Fixed Order (Extended Abstract). In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '94, pages 222–231, New York, NY, USA, 1994. ACM. doi:10.1145/182591.182615.

20 Gerd G. Hillebrand and Paris C. Kanellakis. On the Expressive Power of Simply Typed and Let-Polymorphic Lambda Calculi. In *LICS'96*, 1996.

21 Gerd G. Hillebrand, Paris C. Kanellakis, and Harry G. Mairson. Database Query Languages Embedded in the Typed Lambda Calculus. *Information and Computation*, 127(2):117–144, June 1996. doi:10.1006/inco.1996.0055.

22 Neil Immerman. Languages that Capture Complexity Classes. *SIAM Journal on Computing*, 16(4):760–778, August 1987. doi:10.1137/0216051.

23 Neil Immerman. Nondeterministic Space is Closed under Complementation. *SIAM Journal on Computing*, 17(5):935–938, October 1988. doi:10.1137/0217058.

24 Donald E. Knuth. Mathematics and computer science: Coping with finiteness. *Science*, 194(4271):1235–1242, 1976. doi:10.1126/science.194.4271.1235.

25 Lars Kristiansen. Higher Types, Finite Domains and Resource-bounded Turing Machines. *Journal of Logic and Computation*, 22(2):281–304, April 2012. doi:10.1093/logcom/exq009.

26 Daniel Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 460–469, Tucson, AZ, USA, November 1983. doi:10.1109/SFCS.1983.50.

27 Harry G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, September 1992. doi:10.1016/0304-3975(92)90020-G.

28 Damiano Mazza. Simple Parsimonious Types and Logarithmic Space. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24–40, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2015.24.

29 Damiano Mazza and Kazushige Terui. Parsimonious Types and Non-uniform Computation. In *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 350–361. Springer, Berlin, Heidelberg, July 2015. doi:10.1007/978-3-662-47666-6_28.

30 Paul-André Melliès. Sequential algorithms and strongly stable functions. *Theoretical Computer Science*, 343(1):237–281, October 2005. doi:10.1016/j.tcs.2005.05.015.

31 Paul-André Melliès. On dialogue games and coherent strategies. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 540–562, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2013.540.

32 Lê Thành Dũng Nguyễn, Thomas Seiller, Paolo Pistone, and Lorenzo Tortora De Falco. Finite semantics of polymorphism, complexity and the power of type fixpoints. 2019. URL: https://hal.archives-ouvertes.fr/hal-01979009.

33 A. Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *computational complexity*, 21(4):643–670, December 2012. doi:10.1007/s00037-012-0047-3.

34 K. Reinhardt and E. Allender. Making Nondeterminism Unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, January 2000. doi:10.1137/S0097539798339041.

35 Walter L. Ruzzo, Janos Simon, and Martin Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, April 1984. doi:10.1016/0022-0000(84)90066-7.

36 Ulrich Schöpp. Stratified Bounded Affine Logic for Logarithmic Space. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 411–420, July 2007. doi:10.1109/LICS.2007.45.

37 Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81, July 1979. doi:10.1016/0304-3975(79)90007-0.

38 Kazushige Terui. Proof nets and boolean circuits. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 182–191, 2004. doi:10.1109/LICS.2004.1319612.
39 Kazushige Terui. Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus. In *RTA'12*, 2012.

A The sequent calculus of Linear Logic

The formulas of MALL_2 are given by the grammar

$$A, B := X \mid X^{\perp} \mid 1 \mid \perp \mid A \otimes B \mid A \parr B \mid 0 \mid \top \mid A \oplus B \mid A \,\&\, B \mid \forall X.\, A \mid \exists X.\, B$$

where X belongs to a fixed countable set of variables. ELL_2 formulas are given by an extension of the previous grammar with the exponential modalities !/?.

$$A, B := \cdots \mid !A \mid ?A$$

Customary notations for duality and linear implication are recalled in Figure 3 and the deduction rules for MALL_2 one-sided sequents are given in Figure 2. ELL_2 proofs additionally allow for the rules recalled in Figure 1 (in Section 2).

$$(\texttt{ax}\text{-rule})\frac{}{\vdash A, A^{\perp}} \qquad (\text{cut rule})\frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta} \qquad (\text{exchange rule})\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}$$

$$(\otimes\text{-rule})\frac{\vdash \Gamma, A \quad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta} \qquad (\parr\text{-rule})\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \qquad (\perp\text{-rule})\frac{\vdash \Gamma}{\vdash \Gamma, \perp} \qquad (1\text{-rule})\frac{}{\vdash 1}$$

$$(\oplus\text{-rule})\frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus A_2}\text{for } i \in \{1, 2\} \qquad (\&\text{-rule})\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B} \qquad (\top\text{-rule})\frac{}{\vdash \Gamma, \top}$$

$$(\exists\text{-rule})\frac{\vdash \Gamma, A[B/X]}{\vdash \Gamma, \exists X.\, A} \qquad (\forall\text{-rule})\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall X.\, A}\text{for } X \text{ not free in } \Gamma$$

Figure 2 Rules for the MALL_2 sequent calculus (there is no rule for 0).

A proof is called cut-free if there is no occurence of the cut rule. Cut-free proofs of propositional formulas satisfy the subformula property. Therefore, from a quick syntactic analysis, it follows that there are exactly two cut-free proofs $\vdash \texttt{Bool}$.

There is a standard rewriting system for eliminating the cut rules for both MALL_2 and ELL_2, which can be shown to be terminating and confluent up to some natural commutating conversions. The process of computing a normal form is called *cut-elimination*. We recall in Figure 4 the key reductions involved.

B Proof of Theorem 14 (encoding queries in ELL_2)

We sketch the compilation scheme behind Theorem 14, which enables to go from FO+DTC formulas to $\mathrm{ELL}_2^{\mathrm{PP3}}$ proofs. The construction is done by recursing on the formula φ of interest; assuming k free variables, we map φ to a proof $\pi_\varphi : \texttt{Inp}_\Sigma[\delta] \multimap !!\texttt{Rel}_k[\delta]$. Thus, we have the following cases:

$$
\begin{array}{llll}
1^\perp := \bot & \bot^\perp := 1 & (\exists X.\, A)^\perp := \forall X.\, A^\perp \\
(A \otimes B)^\perp := A^\perp \parr B^\perp & (A \parr B)^\perp := A^\perp \otimes B^\perp & (\forall X.\, A)^\perp := \exists X.\, A^\perp \\
0^\perp := \top & \top^\perp := 0 & (!A)^\perp := ?A^\perp \\
(A \oplus B)^\perp := A^\perp \mathbin{\&} B^\perp & (A \mathbin{\&} B)^\perp := A^\perp \oplus B^\perp & (?A)^\perp := !A^\perp \\
 & A \multimap B := A^\perp \parr B &
\end{array}
$$

Figure 3 Duality for formulas and linear arrow.

$$
\dfrac{\dfrac{}{A, A^\perp} \quad \dfrac{\pi}{A, \Delta}}{A, \Delta} \rightsquigarrow \dfrac{\pi}{A, \Delta}
$$

$$
\dfrac{\dfrac{\pi_1}{\vdash \Gamma, A} \quad \dfrac{\pi_2}{\vdash \Gamma', B}}{\vdash \Gamma, \Gamma', A \otimes B} \quad \dfrac{\dfrac{\pi_3}{\vdash A^\perp, B^\perp, \Delta}}{\vdash A^\perp \parr B^\perp, \Delta}}{\vdash \Gamma, \Gamma', \Delta} \rightsquigarrow \dfrac{\dfrac{\pi_1}{\vdash \Gamma, A} \quad \dfrac{\dfrac{\pi_2}{\vdash \Gamma', B} \quad \dfrac{\pi_3}{\vdash B^\perp, A^\perp, \Delta}}{\vdash A^\perp, \Gamma', \Delta}}{\vdash \Gamma, \Gamma', \Delta}
$$

$$
\dfrac{\dfrac{\dfrac{\pi_1}{\vdash \Gamma, A}}{\vdash \Gamma, A \oplus B} \quad \dfrac{\dfrac{\pi_2}{\vdash A^\perp, \Delta} \quad \dfrac{\pi_3}{\vdash B^\perp, \Delta}}{\vdash A^\perp \mathbin{\&} B^\perp, \Delta}}{\vdash \Gamma, \Delta} \rightsquigarrow \dfrac{\dfrac{\pi_1}{\vdash \Gamma, A} \quad \dfrac{\pi_2}{\vdash A^\perp, \Delta}}{\vdash \Gamma, \Delta}
$$

$$
\dfrac{\dfrac{\dfrac{\pi_1}{\vdash \Gamma, A}}{\vdash ?\Gamma, !A} \quad \dfrac{\dfrac{\pi_2}{\vdash A^\perp, \Delta, B}}{\vdash ?A^\perp, ?\Delta, !B}}{\vdash ?\Gamma, ?\Delta, !B} \rightsquigarrow \dfrac{\dfrac{\pi_1}{\vdash \Gamma, A} \quad \dfrac{\pi_2}{\vdash A^\perp, \Delta, B}}{\dfrac{\vdash \Gamma, \Delta, B}{\vdash ?\Gamma, ?\Delta, !B}}
$$

$$
\dfrac{\dfrac{\pi_1}{\vdash ?\Gamma, !A} \quad \dfrac{\dfrac{\pi_2}{\vdash \Delta}}{\vdash ?A^\perp, \Delta}}{\vdash ?\Gamma, \Delta} \rightsquigarrow \dfrac{\dfrac{\pi_2}{\vdash \Delta}}{\vdash ?\Gamma, \Delta}
$$

$$
\dfrac{\dfrac{\pi_1}{\vdash ?\Gamma, !A} \quad \dfrac{\dfrac{\pi_2}{\vdash ?A^\perp, ?A^\perp, \Delta}}{\vdash ?A^\perp, \Delta}}{\vdash ?\Gamma, \Delta} \rightsquigarrow \dfrac{\dfrac{\pi_1}{\vdash ?\Gamma, !A} \quad \dfrac{\dfrac{\pi_1}{\vdash ?\Gamma, !A} \quad \dfrac{\pi_2}{\vdash ?A^\perp, ?A^\perp, \Delta}}{\vdash ?A^\perp, ?\Gamma, \Delta}}{\dfrac{\vdash ?\Gamma, ?\Gamma, \Delta}{\vdash ?\Gamma, \Delta}}
$$

$$
\dfrac{\dfrac{\dfrac{\pi_1}{\vdash \Gamma, A}}{\vdash \Gamma, \forall X.\, A} \quad \dfrac{\dfrac{\pi_2}{\vdash A^\perp[B/X], \Delta}}{\vdash \exists X.\, A^\perp, \Delta}}{\vdash \Gamma, \Delta} \rightsquigarrow \dfrac{\dfrac{\pi_1[B/X]}{\vdash \Gamma, A[B/X]} \quad \dfrac{\pi_2}{\vdash A^\perp[B/X], \Delta}}{\vdash \Gamma, \Delta}
$$

Figure 4 Key reductions of ELL_2 cut-elimination.

- φ might be a relation $\mathcal{R}_i(\vec{x})$ ($i \in \{0, \dots, N\}$), in which case the $\pi_{\mathcal{R}_i}$ is the composition of the projections $\mathtt{Inp}_\Sigma[\delta] \multimap \mathtt{!!Rel}_{r_i}[\delta]$ and $\delta^k \multimap \delta^{r_i}$ (recall that $\mathtt{Rel}_{r_i}[\delta] = \delta^{r_i} \multimap \mathtt{Bool}$) corresponding respectively to fetching the relation from the finite structure and feeding the appropriate arguments.
- φ can arise from a FO connective ($\exists, \vee$ or $\neg$), in which case, we may cut the proofs corresponding to subformulas with the liftings to relations of $\mathtt{or} : \mathtt{Bool} \otimes \mathtt{Bool} \multimap \mathtt{Bool}$, $\mathtt{not} : \mathtt{Bool} \multimap \mathtt{Bool}$ or $\mathtt{exists} : \mathtt{List}[\delta] \otimes \mathtt{!}(\delta \multimap \mathtt{Bool}) \multimap \mathtt{!Bool}$. Explicit derivations for these proofs in $\mathrm{ELL}_2^{\mathrm{PP3}}$ are given in Figure 7.
- Finally φ could be a DTC operator; this is the most complex case, which we handle separately in Lemma 30.

In the above discussion, save for cuts, all operators syntactically belong to $\mathrm{ELL}_2^{\mathrm{PP3}}$. We can conclude by noticing that cuts along propositional formulas are admissible in $\mathrm{ELL}_2^{\mathrm{PP3}}$ (Proposition 9).

Before turning to the missing link, i.e., the proof of Lemma 30, let us briefly reflect on the computational power available in $\mathrm{ELL}_2^{\mathrm{PP3}}$ proofs of $\mathtt{List}[\delta] \multimap A$. The impredicative $\mathtt{List}[\delta]$ type allows for a straightforward iteration over the list akin to the `fold` operation of functional programming languages or a single `for` loop. It is easy to show that the exponential modality allow to chain such loops, but it is slightly more subtle to check that one can nest them. We require such nesting to implement `dtc`, and we thus show, writing $\mathtt{ctx}(\mathfrak{D})$ for the $\mathtt{Ctx}[\delta]$ appearing in the ELL_2 encoding $\overline{\mathfrak{D}}$ of a finite model:

▶ **Lemma 28.** *For all $k \in \mathbb{N}$, there is a proof $\iota_k : \mathtt{Ctx}[\delta] \vdash \mathtt{!List}[\delta^k]$ such that for all $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$, $\mathbf{cut}(\mathtt{ctx}(\mathfrak{D}), \iota_k[\mathtt{Fin}(n)/\delta])$ reduces to the Church encoding of the list of all elements of D^k, where D is the domain of $\mathfrak{D}$.*

Furthermore, if it is cut with a $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof, $\mathtt{!List}[\delta^k]$ being the cut formula, then the resulting proof reduces to a cut-free proof in $\mathrm{ELL}_2^{\mathrm{PP3}}$.

Proof. Recall that $\mathtt{Ctx}[\delta] = \mathtt{!!W}[\delta] \otimes \mathtt{!!C}[\delta] \otimes \mathtt{!List}[\delta]$. So, equivalently, we are looking for a proof of $\mathtt{!!W}[\delta], \mathtt{!!C}[\delta], \mathtt{!List}[\delta] \vdash \mathtt{!List}[\delta^k]$.

The idea is to start from a proof $\mathtt{combine} : \mathtt{!W}[\delta], \mathtt{!C}[\delta], \mathtt{List}[\delta], \mathtt{List}[\varepsilon] \vdash \mathtt{List}[\delta \otimes \varepsilon]$ implementing a function taking lists $\pi : \mathtt{List}[\delta]$ and $\pi' : \mathtt{List}[\varepsilon]$ returning, after cut-elimination, a list enumerating all pairs of elements from π and π'. We spell out the definition of `combine` in Figure 5 and we leave the checking that it satisfies the above specification to the interested reader. Intuitively, it works as follows: to iterate a $\mathtt{!}(\delta \otimes \varepsilon \multimap Z \multimap Z)$ and obtain a $\mathtt{!}(Z \multimap Z)$,

- first convert it into a $\mathtt{!}(\delta \otimes \varepsilon \multimap Z \multimap \delta \otimes Z)$ by using $\mathtt{!C}[\delta]$ to copy the input
- this being isomorphic to $\mathtt{!}(\varepsilon \multimap \delta \otimes Z \multimap \delta \otimes Z)$, feed it to $\mathtt{List}[\varepsilon]$ to realize an "inner iteration" and get $\mathtt{!}(\delta \otimes Z \multimap \delta \otimes Z)$
- erase the δ on the right-hand side of $\multimap$ via $\mathtt{!W}[\delta]$, and pass the result to $\mathtt{List}[\delta]$.

Clearly, `combine` belongs to $\mathrm{ELL}_2^{\mathrm{PP3}}$ and is cut-free. Furthermore, if it is cut against a $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof, then cut-elimination yields a $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof. To see this, notice that the existential witness of $\mathtt{List}[\delta]^\perp$ and $\mathtt{List}[\varepsilon]^\perp$ (corresponding to the hypotheses $\mathtt{List}[\delta]$ and $\mathtt{List}[\varepsilon]$ on the left of the turnstile) are taken to be the eigenvariable Z of $\mathtt{List}[\delta \otimes \varepsilon]$ and $\delta \otimes Z$ respectively. Therefore, when cut against a $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof $\pi : \mathtt{?List}[\delta \otimes \varepsilon]^\perp, \Gamma$, during cut-elimination, these witnesses may only change when $\mathtt{List}[\delta \otimes \varepsilon]$ is pitted against a $\exists$ rule whose witness A is in PP3. In this case, the witnesses in the $\exists$ rule of `combine` become A and $\delta \otimes A$, which are both still in PP3.

At this stage, it is thus natural to define ι_k for $k \geq 1$ as the proof obtained by normalizing the $\tilde{\iota}_k$ in Figure 6 and adding $\parr$-rules to turn $\mathtt{??C}[\delta]^\perp, \mathtt{??W}[\delta]^\perp, \mathtt{?List}[\delta]^\perp$ into $\mathtt{Ctx}[\delta]^\perp$.

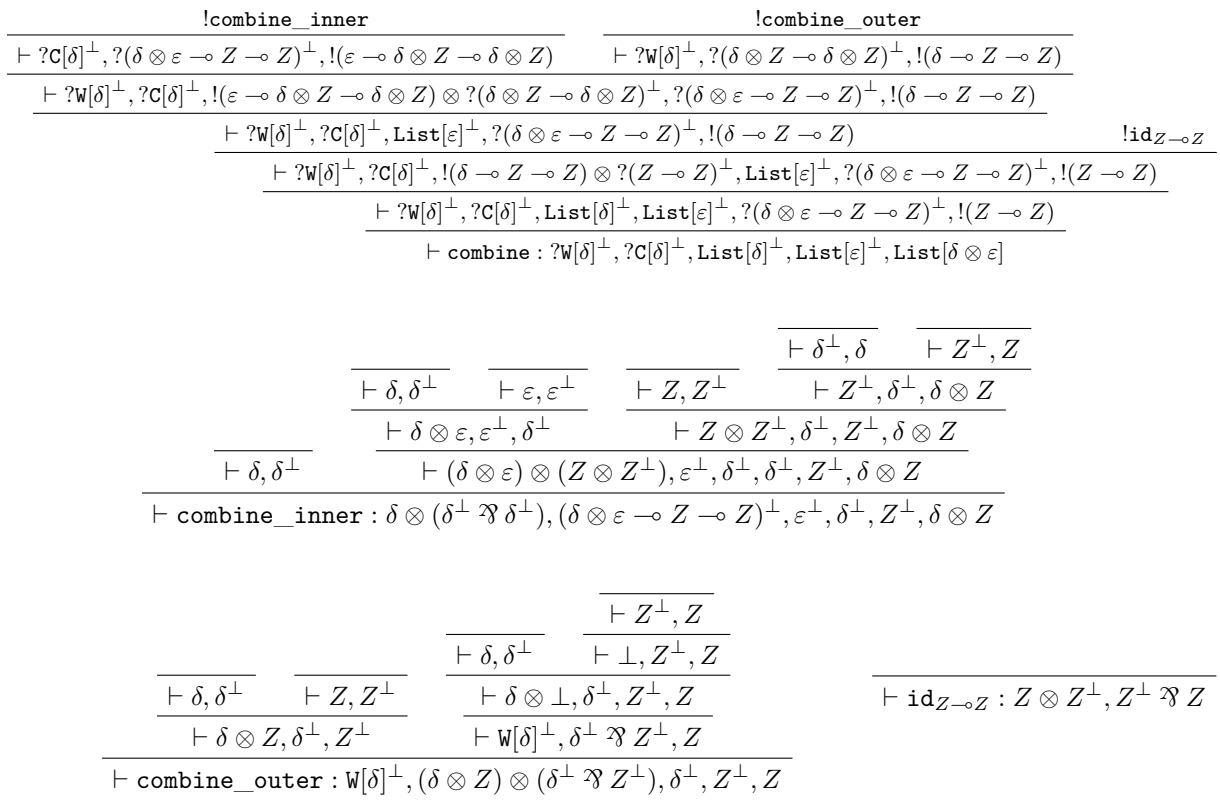


Figure 5 A proof derivation of `combine` (invertible and exchange rules are systematically omitted).

Formally showing that the cutting with an arbitrary proof $\pi : ?\mathtt{List}[\delta^k]^\perp, \Gamma$ and performing cut-elimination still yields an $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof can be done straightforwardly by induction over k by exploiting confluence of cut-elimination and $\tilde{\iota}_k$. ◀

▶ Remark 29. If we replaced PP3 with the class PP2 of formulas $P \multimap R$ with $P, R \in \mathrm{PP}$, then the statement would no longer be true as $\delta \otimes (P \multimap R)$ would not be necessarily PP2. So while the definition of dtc that we give next seems to use only iteration at PP2 types, we are actually forced to go to PP3 to iterate over tuples of arity $k > 1$.

We now turn to sketching the implementation of the DTC operator in $\mathrm{ELL}_2^{\mathrm{PP3}}$.

▶ **Lemma 30.** *For all k, l there exists an $\mathrm{ELL}_2^{\mathrm{PP3}}$ proof $\mathsf{dtc}_{k,l}$ of*

$$\mathtt{Ctx}[\delta], !!\mathtt{Rel}_2[\delta], !!\mathtt{Rel}_{2k+l}[\delta] \vdash !!\mathtt{Rel}_{2k+l}[\delta]$$

such that, if $\phi(\vec{x}, \vec{y}, \vec{z})$ is a FO+DTC formula with $2k+l$ free variables ($|\vec{x}| = |\vec{y}| = k$, $|\vec{z}| = l$), and the relation it defines over $\mathfrak{D} \in \mathrm{FinMod}(\Sigma)$ is encoded as $\overline{\phi} : \mathtt{Rel}_{2k+l}[\mathtt{Fin}(n)]$, then $\mathbf{cut}(\mathsf{ctx}(\mathfrak{D}) \otimes !!\overline{\phi} \otimes !!\overline{\leq^{\mathfrak{D}}}, \mathsf{dtc}_{k,l}[\mathtt{Fin}(n)/\delta])$ reduces to $!!\rho$ where ρ encodes the relation defined by $\mathrm{DTC}_{\vec{x},\vec{y}}(\phi)$ over $\mathfrak{D}$. (So the $!!\mathtt{Rel}_2[\delta]$ is the total order.)

Proof sketch. First, we define an ELL_2 proof $\mathsf{dt}_{k,l}$ of $\mathtt{Ctx}[\delta], !!\mathtt{Rel}_{2k+l}[\delta] \vdash !!(\delta^l \otimes \delta^k \multimap 1 \oplus \delta^k)$ whose role is to compute the partial function associated to the determinization ϕ_d of the input relation ϕ.

$$\dfrac{\dfrac{}{\vdash (!\mathtt{List}[\delta])^{\perp}, !\mathtt{List}[\delta]}}{\vdash \tilde{\iota}_1 : ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, (!\mathtt{List}[\delta])^{\perp}, !\mathtt{List}[\delta]}$$

$$\dfrac{\dfrac{\dfrac{\tilde{\iota}_k}{\vdash ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, ?\mathtt{List}[\delta]^{\perp}, !\mathtt{List}[\delta^k \otimes \delta]} \quad \dfrac{!\mathtt{combine}}{\vdash ?\mathtt{List}[\delta^k]^{\perp}, ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, ?\mathtt{List}[\delta]^{\perp}, !\mathtt{List}[\delta^k \otimes \delta]}}{\vdash ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, ?\mathtt{List}[\delta]^{\perp}, ?\mathtt{List}[\delta]^{\perp} !\mathtt{List}[\delta^k \otimes \delta]}}{\vdash \tilde{\iota}_{k+1} : ??\mathtt{C}[\delta]^{\perp}, ??\mathtt{W}[\delta]^{\perp}, ?\mathtt{List}[\delta]^{\perp}, !\mathtt{List}[\delta^k \otimes \delta]}$$

Figure 6 Derivations of $\tilde{\iota}_k$.

$$\dfrac{\dfrac{}{\vdash 1_l}}{\vdash \mathtt{true} : 1_l \oplus 1_r} \qquad \dfrac{\dfrac{}{\vdash 1_r}}{\vdash \mathtt{false} : 1_l \oplus 1_r} \qquad \dfrac{\dfrac{\dfrac{\mathtt{false}}{\vdash \mathtt{Bool}}}{\vdash \perp_l, \mathtt{Bool}} \quad \dfrac{\dfrac{\mathtt{true}}{\vdash \mathtt{Bool}}}{\vdash \perp_r, \mathtt{Bool}}}{\vdash \mathtt{neg} : \perp_l \,\&\, \perp_r, \mathtt{Bool}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{true}}{\vdash \mathtt{Bool}}}{\vdash \perp_{l_1}, \perp_{l_2}, \mathtt{Bool}} \quad \dfrac{\dfrac{\mathtt{true}}{\vdash \mathtt{Bool}}}{\vdash \perp_{l_1}, \perp_{r_2}, \mathtt{Bool}}}{\vdash \perp_{l_1}, \perp_{l_2} \,\&\, \perp_{r_2}, \mathtt{Bool}} \quad \dfrac{\dfrac{\dfrac{\mathtt{true}}{\vdash \mathtt{Bool}}}{\vdash \perp_{r_1}, \perp_{l_2}, \mathtt{Bool}} \quad \dfrac{\dfrac{\mathtt{false}}{\vdash \mathtt{Bool}}}{\vdash \perp_{r_1}, \perp_{r_2}, \mathtt{Bool}}}{\vdash \perp_{r_1}, \perp_{l_2} \,\&\, \perp_{r_2}, \mathtt{Bool}}}{\vdash \mathtt{or} : \perp_{l_1} \,\&\, \perp_{r_1}, \perp_{l_2} \,\&\, \perp_{r_2}, \mathtt{Bool}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \delta, \delta^{\perp}} \quad \dfrac{\mathtt{or}}{\vdash \mathtt{Bool}^{\perp}, \mathtt{Bool}^{\perp}, \mathtt{Bool}}}{\vdash \delta \otimes \mathtt{Bool}^{\perp}, \delta^{\perp}, \mathtt{Bool}^{\perp}, \mathtt{Bool}}}{\vdash ?(\delta \multimap \mathtt{Bool})^{\perp}, !(\delta \multimap \mathtt{Bool} \multimap \mathtt{Bool})} \quad \dfrac{\dfrac{\dfrac{\mathtt{false}}{\vdash \mathtt{Bool}} \quad \dfrac{}{\vdash \mathtt{Bool}^{\perp}, \mathtt{Bool}}}{\vdash \mathtt{Bool} \otimes \mathtt{Bool}^{\perp}, \mathtt{Bool}}}{\vdash ?(\mathtt{Bool} \multimap \mathtt{Bool})^{\perp}, !\mathtt{Bool}}}{\vdash !(\delta \multimap \mathtt{Bool} \multimap \mathtt{Bool}) \otimes ?(\mathtt{Bool} \multimap \mathtt{Bool})^{\perp}, ?(\delta \multimap \mathtt{Bool})^{\perp}, !\mathtt{Bool}}$$

$$\vdash \mathtt{exists} : \mathtt{List}[\delta]^{\perp}, ?(\delta \multimap \mathtt{Bool})^{\perp}, !\mathtt{Bool}$$

Figure 7 Encoding of FO connectives as $\mathrm{ELL}_2^{\mathrm{PP3}}$ proofs.

In fact we will first output the type $!(\delta^l \otimes \delta^k \multimap \mathsf{M}(\delta^k))$ where $\mathsf{M}(A) = 1 \oplus A \oplus 1$ should be thought of as the algebraic data type $\mathsf{M}(A) := \mathsf{None} \mid \mathsf{Unique}\, A \mid \mathsf{Multiple}$; then we post-compose with $1 \oplus \delta^k \oplus 1 \multimap 1 \oplus \delta^k$ which sends both None and $\mathsf{Multiple}$ to $\mathsf{inl}(1)$.

$\mathsf{dt}_{k,l}$ is built by iterating a function of type $\delta^k \multimap (\delta^l \otimes \delta^k \multimap \mathsf{M}(\delta^k)) \multimap (\delta^l \otimes \delta^k \multimap \mathsf{M}(\delta^k))$ over all possible k-tuples of domain elements as first argument, starting from the constant function $\lambda(\_ : \delta^l \otimes \delta^k).\, \mathsf{None}$ (which can be defined thanks to $\mathtt{W}[\delta]$), using Lemma 28. The iterated function is described by the following functional pseudocode, where $r : \mathtt{Rel}_{2k+l}[\delta] = \delta^k \otimes \delta^k \otimes \delta^l \multimap \mathtt{Bool}$ is one of the arguments of $\mathsf{dt}_{k,l}$:

- $\lambda\vec{a}.\lambda f.\lambda(\vec{p} \otimes \vec{b} : \delta^l \otimes \delta^k).\, \mathsf{if}\ r\ (\vec{b} \otimes \vec{a} \otimes \vec{p})$
  - $\mathsf{then\ case}\ f\ (\vec{p} \otimes \vec{b})\ \mathsf{of}$
    - $\mathsf{None} \to \mathsf{Unique}\, \vec{a}$
    - $\mathsf{Unique}\, \vec{c} \to \mathsf{Multiple}$
    - $\mathsf{Multiple} \to \mathsf{Multiple}$
  - $\mathsf{else}\ f\ (\vec{p} \otimes \vec{b})$

(using $\mathsf{if/then/else}$ to destruct $\mathtt{Bool}$). Note that $\vec{a}$, $\vec{b}$, $\vec{c}$ and $\vec{p}$ are used non-linearly; this means we have to insert the appropriate calls to $\mathtt{C}[\delta]$ and $\mathtt{W}[\delta]$ to get a proper ELL_2 proof.

After that, as sketched before, we need to compute a least fixpoint by iterating $n^{2k}+1$ times some functional which uses the partial function produced by $\mathsf{dt}_{k,l}$. To do so, we first cast $\mathtt{List}[\delta]$ into $\mathtt{Nat} = \forall X.\, !(X \multimap X) \multimap !(X \multimap X)$ thanks to $\mathtt{W}[\delta]$; then the polynomial $P(n) = n^{2k}+1$ can be defined as a proof $!\mathtt{Nat} \vdash !\mathtt{Nat}$ (see [8]). The function to be iterated $n^{2k}+1$ times is of type $\mathtt{Rel}_{2k+l}[\delta] \multimap \mathtt{Rel}_{2k+l}[\delta]$ and is defined as follows, where f is the output of $\mathsf{dt}_{k,l}$:

- $\lambda q.\lambda(\vec{a} \otimes \vec{b} \otimes \vec{p} : \delta^k \otimes \delta^k \otimes \delta^l).\, \mathsf{case}\ f\ (\vec{p} \otimes \vec{a})\ \mathsf{of}$
  - $\mathsf{inl}(1) \to \mathsf{if}\ q\ (\vec{a} \otimes \vec{b} \otimes \vec{p})\ \mathsf{then}\ \vec{a} = \vec{b}\ \mathsf{else}\ \vec{a} = \vec{b}$
  - $\mathsf{inr}(\vec{c}) \to \vec{a} = \vec{b} \mid\mid q\ (\vec{c} \otimes \vec{b} \otimes \vec{p})$

This uses an equality predicate $=$, which is derived easily from the total order relation (this is why we need $\mathtt{Rel}_2[\delta]$ in the input), and the disjunction $||$ on the booleans $\mathtt{Bool}$ which is given by the proof $\mathtt{or}$ in Figure 7. Note also that in the $\mathsf{inl}(1)$ branch, the $\mathsf{if/then/else}$ is only used to throw away the result of q, because the function has to be linear in q.

Thus we get a function $\mathtt{Rel}_{2k+l}[\delta] \multimap \mathtt{Rel}_{2k+l}[\delta]$ corresponding to the $\psi_R^{n^{2k}+1}$ earlier; applying this to the argument $\lambda(\_ : \delta^{2k+l}).\, \mathsf{false}$, which represents the empty relation $\emptyset$, yields the deterministic transitive closure of the input. ◀

C Proof of Proposition 17 (syntactic analysis)

We start by an analysis mirroring that of [32, Lemma III.5], using:

- the reversibility of the rules for $\forall$ and $⅋$,
- the commutation of $\exists$-intro with all rules but $\forall$-intro,
- the fact that functorial promotion is the only way to introduce $!$,

to ensure that a proof $\pi : \forall\delta.((!\mathtt{List}[\delta] \otimes !!A_1 \otimes \ldots \otimes !!A_k) \multimap !!B)$ must be, modulo commutative conversions, of the form (where a double horizontal line indicates a sequence of inference rules)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\hat{\pi}}{\vdash \texttt{List}[\delta][U_1]^\perp, \dots, \texttt{List}[\delta][U_m]^\perp, ?\Gamma, !B}}{\vdash \texttt{List}[\delta][U_1]^\perp, \dots, \texttt{List}[\delta][U_m]^\perp, ?\Delta, !B}\,\text{C/W}}{\vdash ?\texttt{List}[\delta][U_1]^\perp, \dots, ?\texttt{List}[\delta][U_m]^\perp, ??\Delta, !!B}}{\vdash ?\texttt{List}[\delta]^\perp, ??A_1^\perp, \dots, ??A_k^\perp, !!B}\,\text{C/W}}{\vdash \forall\delta.((!\texttt{List}[\delta] \otimes !!A_1 \otimes \dots \otimes !!A_k) \multimap !!B)}$$

where

- $\texttt{List}[\delta][U] = !(\delta \multimap U \multimap U) \multimap !(U \multimap U)$ so that $\texttt{List}[\delta] = \forall X.\, \texttt{List}[\delta][X]$
- Γ and Δ are multisets whose supports are included in $\{A_1, \dots, A_k\}$
- the double lines labeled C/W consist in sequences of contractions and weakenings.

We may further assume that we have commuted all contractions/weakenings on $?\Gamma$ downwards in π (this is always possible in the absence of &-rules, and this absence is guaranteed by the subformula property). So we also require that $\hat{\pi}$ does not contain structural rules applied on $?\Gamma$ – equivalently, all the formulas of $!\Gamma$ are introduced by functorial promotion.

Without loss of generality, we can also consider that $U_1, \dots, U_m$ are MALL_2 types, by the same "collapse at depth 2" argument used in the prequel [32, Lemma III.6].

Finally, we have reduced the proposition we wanted to prove to the following lemma. (When $\Gamma = A_1, \dots, A_k$, we write $?\Gamma$ for $?A_1, \dots, ?A_k$ (as in Figure 1) and $\mathcal{C}(\Gamma; B)$ for $\mathcal{C}(A_1, \dots, A_k; B)$ (cf. Definition 16).)

▶ **Lemma 31.** *Let $\hat{\pi}$ be a proof of $\vdash ?\Gamma, \texttt{List}[\delta][U_1]^\perp, \dots, \texttt{List}[\delta][U_m]^\perp, !B$ ($m \geq 0$) where $U_1, \dots, U_m$ and the formulas in Γ are in MALL_2 and have at most one type variable δ. Suppose that $\hat{\pi}$ does not contain structural rules applied to $?\Gamma$.*

Then there exists $f_{\hat{\pi}} \in \mathcal{C}(\Gamma; B)$ such that, for all MALL_2 types T, lists L of proofs of T and proofs $\rho_i : A_i$ where $\Gamma = A_1, \dots, A_k$, we have[15]

$$\mathbf{cut}(!\rho_1 \otimes \dots \otimes !\rho_k \otimes L^{U_1} \otimes \dots \otimes L^{U_m}, \hat{\pi}[T/\delta]) = !f_{\hat{\pi}}(T; L; \rho_1, \dots, \rho_k)$$

where $L^U : !(T \multimap U \multimap U) \multimap !(U \multimap U)$ is the instantiation at U of the ELL_2 Church encoding of the list L.

Proof. By strong induction on m.

In the case $m = 0$, since we assumed that no structural rule are performed, the last rule of $\hat{\pi}$ must introduce $!B$, so $\hat{\pi}$ must come from a functorial promotion applied to a MALL_2 proof $\tilde{\pi}$ of $\vdash \Gamma, B$. Then we obtain $f_{\hat{\pi}}$ by the base case of Definition 16 applied to $\tilde{\pi}$.

If $m \geq 1$, the last rule cannot introduce $!B$ because the context is not of the form $?\Delta$, so it must introduce one of the $\texttt{List}[\delta][U_i]^\perp = !(T \multimap U_i \multimap U_i) \otimes ?(U_i \otimes U_i^\perp)$ whose head connective is $\otimes$. This means that up to commuting contraction downwards, $\hat{\pi}$ must have the shape

$$\dfrac{\dfrac{\hat{\pi}'}{\vdash ?\Gamma', \Delta', !(T \multimap V \multimap V)} \qquad \dfrac{\dfrac{\hat{\pi}''}{\vdash ?(V \otimes V^\perp), \dots, ?(V \otimes V^\perp), ?\Gamma'', \Delta'', !B}}{\vdash ?(V \otimes V^\perp), ?\Gamma'', \Delta'', !B}}{\vdash ?\Gamma, \texttt{List}[\delta][U_1]^\perp, \dots, \texttt{List}[\delta][U_m]^\perp, !B}$$

[15] With a slight abuse of notation: we write $\mathbf{cut}(x_1 \otimes \dots \otimes x_l, y)$ instead of $\mathbf{cut}(x_1, \dots \mathbf{cut}(x_l, y) \dots)$.

where $\Gamma = \Gamma' \cup \Gamma''$, $\Delta' = \mathtt{List}[\delta][U'_1], \dots, \mathtt{List}[\delta][U'_{m'}]$, $\Delta'' = \mathtt{List}[\delta][U'_1], \dots, \mathtt{List}[\delta][U''_{m''}]$, $\{U_1, \dots, U_m\} \setminus \{V\} = \{U'_1, \dots, U'_{m'}\} \cup \{U''_1, \dots, U''_{m''}\}$ and $\hat{\pi}'$, $\hat{\pi}''$ do not use structural rules on either $?\Gamma'$ or $?(V \otimes V^\perp), \dots, ?(V \otimes V^\perp), ?\Gamma''$.

So $m', m'' < m$ and we may apply the induction hypothesis to obtain the functions $f' = f_{\hat{\pi}'} \in \mathcal{C}(\Gamma'; T \multimap V \multimap V)$ and $f'' = f_{\hat{\pi}''} \in \mathcal{C}(V \multimap V, \dots, V \multimap V, \Gamma''; B)$. We apply the iteration scheme to f' to obtain $g \in \mathcal{C}(\Gamma'; V \multimap V)$. The function we are looking for is then

$$f_{\hat{\pi}}(T; L; \rho_1, \dots, \rho_k) = f''(T; L; g(\rho_{i_1}, \dots, \rho_{i_{m'}}), \dots, g(\rho_{i_1}, \dots, \rho_{i_{m'}}), \rho_{j_1}, \dots, \rho_{j_{m''}})$$

which is in $\mathcal{C}(\Gamma; B)$ thanks to the composition scheme and the projections. ◀