A feasible and unitary quantum programming language

Alejandro Díaz-Caro, Emmanuel Hainry, Romain Péchoux, Mário Silva

A feasible and unitary quantum programming language

ALEJANDRO DÍAZ-CARO, Universidad de Buenos Aires-CONICET, ICC, Buenos Aires, Argentina and Universidad Nacional de Quilmes, DCyT, Bernal, Argentina
EMMANUEL HAINRY, ROMAIN PÉCHOUX, and MÁRIO SILVA, Université de Lorraine, CNRS, Inria, LORIA, F54000 Nancy, France

We introduce a novel quantum programming language featuring higher-order programs and quantum control flow which ensures that all qubit transformations are unitary. Our language boasts a type system guaranteeing both unitarity and polynomial-time normalization. Unitarity is achieved by using a special modality for superpositions while requiring orthogonality among superposed terms. Polynomial-time normalization is achieved using a linear-logic-based type discipline employing Barber and Plotkin duality along with a specific modality to account for potential duplications. This type discipline also guarantees that derived values have polynomial size. Our language seamlessly combines the two modalities: quantum circuit programs uphold unitarity, and all programs are evaluated in polynomial time, ensuring their feasibility.

ACM Reference Format:
Alejandro Díaz-Caro, Emmanuel Hainry, Romain Péchoux, and Mário Silva. 2024. A feasible and unitary quantum programming language. 1, 1 (March 2024), 47 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Motivation

Classical control vs quantum control. Quantum programming languages can be classified into two primary categories based on their control flow handling. On the one hand, *classical control* [Selinger 2004] involves quantum operations executed on a specialized device within a classical computer. The classical computer manages program execution by instructing which quantum operation to apply to specific qubits. This approach resembles circuit description languages that use high-level operations on quantum circuits, with examples such as the quantum lambda calculus [Selinger and Valiron 2006], Quipper [Green et al. 2013], and Qwire [Paykin et al. 2017]. Ensuring physical implementability typically involves constraints and linearity-based type systems on quantum data to maintain essential quantum physics properties such as the no-cloning theorem [Wooters and Zurek 1982]. Notably, this category accommodates models like QRAM [Knill 1996].

On the other hand, *quantum control* [Díaz-Caro 2021] allows programming quantum operations based on quantum data, a fundamental concept in quantum computing. For example, the CNOT operation governs the application of a NOT operation based on a control qubit. More advanced examples like the Quantum Switch [Procopio et al. 2015; Rubino et al. 2017] have gained popularity, enabling the application of two operations in different orders based on a control qubit.

While both categories offer equivalent computational power, quantum control provides a more natural approach, allowing programs to be "fully quantum" with operational control flow over

Partially funded by the French-Argentinian IRP SINFIN. Díaz-Caro is also funded by PICT 2021-I-A-00090 and 2019-1272, and PIP 11220200100368CO.

ACM XXXX-XXXX/2024/3-ART
https://doi.org/XXXXXXX.XXXXXXX

quantum data. Programming languages like QML [Altenkirch and Grattage 2005], Lambda-$\mathcal{S}_1$ [Díaz-Caro and 2022], and Qunity [Voichick et al. 2023] fall into this category. These programs must overcome significant constraints to ensure physical implementability, particularly in terms of efficient compilation into quantum circuits or low-level models like QRAM.

Unitarity. One major issue in quantum control is ensuring *unitarity*, a fundamental property of quantum systems that maintains the total probability of all possible outcomes over time. Quantum gates in circuits are represented by unitary maps, preserving ℓ^2-norm and orthogonality. However, representing quantum data as linear combinations in a Hilbert space leads to a problem: programs must have an ℓ^2-norm of 1 (i.e., must lie in the unitary sphere) to be physically realizable, often requiring orthogonality among program branches. This property is generally not preserved by program semantics, leading to the need to restrict programs to those satisfying unitarity. The question of ensuring this restriction through type systems has been first explored by [Altenkirch and Grattage 2005] to some extent, and more recently by [Díaz-Caro et al. 2019b], characterizing superpositions and isometries with realizability techniques [Miquel 2011; Van Oosten 2008]. In this realizability model, types are interpreted as subsets of a vector space's unit sphere, with all typed terms preserving the ℓ^2-norm, and quantum data expressed as superpositions of classical data types. The corresponding typing discipline, that is in a way the dual to Intuitionistic Linear Logic [Girard and Lafont 1987; Hyland and de Paiva 1993], has been introduced in [Díaz-Caro and Malherbe 2022] to delineate a programming language for unitarity called Lambda-$\mathcal{S}_1$.

Feasibility. While unitarity is essential for quantum program implementability, it is insufficient. *Feasibility* [Gurevich 1983] (or *tractability*), the property that a program can be executed within reasonable time and space-constraints, is equally crucial. Compiling quantum programs to low-level models, like quantum circuits, requires imposing restrictions on qubit count, gate count, and error rates. Achieving feasibility involves studying program computational complexity, typically related to polynomial-depth uniform circuit families. Yao's Theorem [Yao 1993] links such families to Bounded-error Quantum Polynomial time (BQP) [Bernstein and Vazirani 1997], a quantum analogue of the probabilistic complexity class BPP. Feasibility has deep roots in classical complexity, leading to fields like descriptive complexity [Immerman 1999] and implicit computational complexity [Péchoux 2020] characterizing complexity classes logically and through programming languages.

Previous work [Dal Lago et al. 2010], based on light linear logic [Girard 1998], characterizes polynomial time in quantum lambda calculus. However, no type system currently ensures quantum program feasibility with quantum control. Thus, a major problem in quantum computing is to develop programming languages with quantum control, typed in such a way as to ensure both unitarity and feasibility of the programmed functions, so that they have a physical implementation that does not break the laws of quantum mechanics, and can (at least in principle) be efficiently compiled into a circuit.

1.2 Contribution

We present PUNQ, a typed quantum programming language with quantum control addressing unitarity and feasibility. Quantum control employs a quantum conditional inspired by QML [Altenkirch and Gratt 2005], producing superposed outputs from superposed inputs. Unitarity is achieved via a variant of Lambda-$\mathcal{S}_1$ [Díaz-Caro and Malherbe 2022], introducing a modality $\sharp$ for superpositions, addressing quantum no-cloning. This modality marks non-duplicable types, treating superpositions of type $\sharp A$ linearly. Only terms of types distinct from $\sharp A$ can be duplicated. For example, $\sharp\mathbb{B}$ denotes superpositions of Booleans (qubits), and realizers of $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$ represent single-qubit

quantum gates. Feasibility is ensured by a variant of the Dual Light Affine Logic type system (DLAL) [Baillot and Terui 2009], employing a modality § to account for potential duplication and a duality à la [Barber and Plotkin 1996], with two arrows corresponding to a non-linear context and a linear context, respectively. In this setting, linear arrows are strictly linear to preserve unitarity and non-linear arrows cannot be applied to superpositions, thus, ensuring the no-cloning principle of quantum mechanics.

The main contributions of this paper are:

- a new typed programming language with quantum control that enjoys subject reduction (Theorem 3.3) and progress (Theorem 3.4),
- a soundness result (Theorem 5.6) showing that typable linear maps over qubits encode isometries and unitary operators when dimensions match,
- a completeness result (Theorem 5.7) stating that any isometry can be encoded by a PUNQ program,
- a non-separability result (Theorem 5.8): there is no linear map that can separate qubits,
- a feasibility result (Theorem 6.12), ensuring polynomial time normalization and polynomially bounded size of normal forms,
- a complexity result for type checking over three fragments of PUNQ (Theorem 7.7). We identify two fragments in which we can check orthogonality in polytime and one in which it is potentially undecidable (Lemma 7.6), and give examples of the expressivity of each fragment.

In summary, PUNQ can be viewed as the first feasible and physically realistic quantum programming language with quantum control. We provide several simple examples to illustrate our results: towards completeness, we show that standard one and two-qubit gates can be simulated by programs (Examples 4.2, 4.3, and 4.4), we provide the encoding of a simple quantum teleportation protocol (Example 4.5) illustrating that soundness can be used to certify unitarity, we also provide a quantum random walk algorithm (Example 4.6) illustrating polynomial time normalization.

1.3 Related Work

Quantum control and unitarity. QML [Altenkirch and Grattage 2005] was the pioneering language to introduce a quantum if-statement, enabling superposition in branches based on the superposition in the if-statement guard. The ℓ^2-norm preservation is ensured through a semantic notion of validity: an if-statement is valid when its branches are orthogonal, effectively reducing the two branches to orthogonal values. Another approach for handling quantum control and superpositions, which is less semantics-driven, is the introduction of Lineal [Arrighi and Dowek 2017]. It is an untyped lambda calculus extended with superpositions of terms. Lineal strictly covers measurement-free quantum programs, as it does not involve orthogonality checks and does not enforce superpositions to have norm 1. However, it treats function application linearly, providing a generalization of the QML if-statement. Lambda-$\mathcal{S}_1$ [Díaz-Caro et al. 2019b; Díaz-Caro and Malherbe 2022], as well as the current work, can be seen as the ℓ^2-norm preserving restriction of Lineal. Lambda-$\mathcal{S}$ [Díaz-Caro et al. 2019a; Díaz-Caro and Malherbe 2023], a preliminary version of this typing discipline, does not ensure unitarity but includes measurements. Qunity [Voichick et al. 2023] enforces the no-cloning property using constrained sharing, allowing duplicated variables to produce only entangled states in a basis-dependent way. While this ensures that well-typed programs in Qunity have a quantum circuit representation, the typing does not guarantee any resource bound, neither on the time complexity of the programs, nor on the size of the compiled circuit. There have also been attempts [Ying 2016] to define a notion of "quantum alternation" allowing measurements to be quantum-controlled, where the resulting system is not monotone

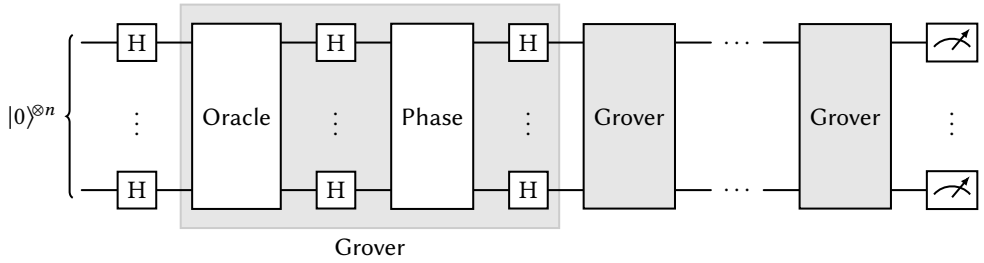


Fig. 1. Quantum search algorithm.

with respect to the Löwner order and, hence, cannot be considered to be a physically feasible concept [Badescu and Panangaden 2015]. An alternative approach in [Yuan et al. 2023] characterizes unitarity using a model based on injective semantics. Complementary approaches following the ZX-calculus research line provide graphical languages for quantum control with quantum tests [Chardonnet 2023; Chardonnet et al. 2022]. Currently, none of the mentioned systems can guarantee the feasibility of their programs in terms of complexity or resource requirements.

Quantum complexity classes. Programming-language-based characterizations of well-known complexity classes have been deeply studied in the field of Implicit Computational Complexity (see [Péchoux 2020] for a survey). To mention a few of them, [Bellantoni and Cook 1992] provided the first implicit (i.e., where the complexity bound does not need to be explicited by the programmer) characterization of polynomial time and [Gaboardi et al. 2008] is the first lambda-calculus characterization of polynomial space. Although a great deal of work has been done in this field over the last three decades, only a small number of it has focused on the quantum paradigm. The paper [Dal Lago et al. 2010] characterizes BQP on the quantum lambda-calculus. However, due to the presence of unrestricted measurement, this classically-controlled system cannot guarantee unitarity. Finally, [Hainry et al. 2023; Yamakami 2020] provide two characterizations of FBQP with quantum control which are restricted to first-order.

1.4 Illustrating Example: Grover's Quantum Search Algorithm

As an example of a PUNQ program, we will consider the algorithm for quantum search [Grover 1996]. We are given a function $f : \{0,1\}^n \to \{0,1\}$ such that, for exactly one input $w \in \{0,1\}^n$, we have that $f(w) = 1$. In the classical case, finding w among $N \triangleq 2^n$ possibilities has average complexity $O(N)$, whereas using Grover's algorithm for quantum search, the value of w can be found with high probability with only $O(\sqrt{N})$ operations (see Figure 1).

Let $\mathbb{B}$ be the type of Booleans. For $n \geq 1$, define the type of tuples of bits as $\mathbb{B}^{n+1} \triangleq \mathbb{B} \times \mathbb{B}^n$ with $\mathbb{B}^1 \triangleq \mathbb{B}$. The type $\sharp\mathbb{B}$ corresponds to qubits and $\sharp(\mathbb{B}^n)$, with $n \geq 1$, is the type of a tuple of n (possibly entangled) qubits. For example, a superposition $|\pm\rangle \triangleq \frac{1}{\sqrt{2}} \cdot |0\rangle \pm \frac{1}{\sqrt{2}} \cdot |1\rangle$ has type $\sharp\mathbb{B}$ and, given a string $x = x_1 \dots x_n \in \{0,1\}^n$, the tensor product state $|x\rangle \triangleq |x_1\rangle \otimes \dots \otimes |x_n\rangle$ can be encoded by a term of type $\sharp(\mathbb{B}^n)$.

We will now describe a PUNQ term encoding the algorithm for Grover search in the case $n = 2$. A very basic component is the Hadamard transformation

$$\mathsf{H} \triangleq \lambda x.\mathsf{if}\ x\ \mathsf{then}\ |+\rangle\,\mathsf{else}\ |-\rangle : \sharp\mathbb{B} \multimap \sharp\mathbb{B},$$

described in more detail in Example 4.2. Intuitively, the type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)$ corresponds to linear maps (unitary gates) over $n \geq 1$ qubits. The if construction assigns the first case to state $|0\rangle$ and

the second to $|1\rangle$. Therefore, for $\alpha, \beta \in \mathbb{C}$, we have the reduction

$$\mathsf{H}\ (\alpha \cdot |0\rangle + \beta \cdot |1\rangle) \rightsquigarrow \alpha \cdot \text{if } |0\rangle \text{ then } |+\rangle \text{ else } |1\rangle\ + \beta \cdot \text{if } |1\rangle \text{ then } |+\rangle \text{else } |1\rangle \rightsquigarrow \alpha \cdot |+\rangle + \beta \cdot |-\rangle$$

We can then easily design a term applying the Hadamard gate to two qubits

$$\mathsf{H}_2 \triangleq \lambda z.\text{let } (x, y) = z \text{ in } (\mathsf{H}\ x, \mathsf{H}\ y) : \sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2).$$

We can likewise define the phase shift operator P, where $P\,|x\rangle \triangleq -|x\rangle$for all $x \neq 0^n$ and $P\,|0^n\rangle \triangleq |0^n\rangle$. For $n = 2$, we can encode P by the term Phase : $\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2)$:

$$\mathsf{Phase} \triangleq \lambda z.\text{let } (x, y) = z \text{ in if } x \text{ then } \big(\text{if } y \text{ then } (|0\rangle, |0\rangle) \text{ else } -1 \cdot (|0\rangle, |1\rangle)\big) \text{ else } -1 \cdot (|1\rangle, y)$$

We will consider an oracle for f given by a unitary O where $O|x\rangle \triangleq (-1)^{f(x)}|x\rangle$, for all $x \in \{0,1\}^n$, i.e., where the oracle acts as the identity for all basis states except for w, on which it performs a phase shift of -1. This represents a unitary gate of dimension n, and therefore there exists a PUNQ term, say Oracle, with type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)$, that encodes it. We may now define the Grover iteration step in the algorithm:

$$\mathsf{Grover} \triangleq \lambda x.\mathsf{H}_2\ (\mathsf{Phase}\ (\mathsf{H}_2\ (\mathsf{Oracle}\ x))) : \sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2).$$

In order to iterate the Grover step, we use the Church numeral encoding $\underline{n} \triangleq \lambda f.\lambda x.f^n(x)$ with type $\mathbb{N} \triangleq \forall X.(X \multimap X) \Rightarrow \S(X \multimap X)$. The type of Church numerals makes use of two extra constructs: a non-linear arrow $\Rightarrow$, whose input cannot be quantum data (superpositions), so that PUNQ programs preserve the laws of quantum mechanics (e.g. no-cloning); a modality § from Girard's light linear logic [Girard 1998], accounting for possible duplication in a term.

This term can be applied to our term Grover with the substitution $X = \sharp(\mathbb{B}^2)$. Notice that it does not break no-cloning as it roughly corresponds to a term of type $(\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2)) \Rightarrow \S(\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2))$, whose input can be duplicated. Indeed, the input here is a quantum gate of type $(\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2))$ which does not constitute quantum data.

As such, we obtain the final term:

$$\mathsf{Search} \triangleq \lambda m.\lambda x.(m\ \mathsf{Grover})(\mathsf{H}_2\ x) : \mathbb{N} \multimap \S\sharp(\mathbb{B}^2) \multimap \S\sharp(\mathbb{B}^2).$$

The term Search $\underline{m} : \S\sharp(\mathbb{B}^2) \multimap \S\sharp(\mathbb{B}^2)$ simulates a precise run of m iterations and produces as output a possibly entangled two-qubit state – which is indeed the output before we perform any measurements.

The non-trivial properties ensured by our type discipline on the above term are the following:

- By Theorem 5.6, for any given integer m, since the term Search $\underline{m}$ has type $\S\sharp(\mathbb{B}^2) \multimap \S\sharp(\mathbb{B}^2)$, it represents a unitary transformation with algebraic coefficients (we will later put a restriction on complex numbers to avoid the consideration of non-computable numbers), which can therefore be physically implemented in a quantum circuit, such as Figure 1.
- By Theorem 6.12 (*Polynomial time normalization*) the PUNQ type discipline ensures that the typed term Search $\underline{m}\ (|0\rangle, |0\rangle) : \S\sharp(\mathbb{B}^2)$ reduces to a normal form $\sum_{i,j\in\{0,1\}} \alpha_{ij} \cdot (|i\rangle, |j\rangle)$ in a number of steps that is polynomial on the original size of the term (in this case, the number of Grover iterations is linear on the parameter m given – for a more elaborate iteration example, see Example 4.6). Notice that this property remains valid for any (possibly entangled) pair encoding a two-qubit state given as input.

2 A PROGRAMMING LANGUAGE WITH QUANTUM CONTROL

2.1 Syntax

PUNQ (short for Polytime UNitary Quantum language) is a programming language with syntax defined by the grammar in Figure 2. A *term* can take the form of a variable x, a bit $|0\rangle$ or $|1\rangle$,

$$
\begin{array}{rl}
\text{(Terms)} & \mathsf{T} \ni t := x \mid |0\rangle \mid |1\rangle \mid \text{if } t \text{ then } \vec{t} \text{ else } \vec{t} \mid \lambda x.\vec{t} \mid t\ t \mid (t,t) \mid \text{let } (x,y) = t \text{ in } \vec{t} \\
\text{(Superpositions)} & \mathsf{S} \ni \vec{t} := t \mid \vec{0} \mid \alpha \cdot \vec{t} \mid \vec{t} + \vec{t} \\
\text{(Basis values)} & \mathsf{BV} \ni v := |0\rangle \mid |1\rangle \mid \lambda x.\vec{t} \mid (v,v) \\
\text{(Values)} & \mathsf{V} \ni \vec{v} := v \mid \vec{0} \mid \alpha \cdot \vec{v} \mid \vec{v} + \vec{v}
\end{array}
$$

Fig. 2. Syntax of PUNQ programs.

a conditional statement, an abstraction, an application, a pair, or a pair destructor. We denote the set of terms as T, and the terms are denoted by r, s, t_1, t_2, and so on. A *superposition* can be either a term t, the null vector $\vec{0}$, the product $\alpha \cdot \vec{t}$ of a superposition $\vec{t}$ with an algebraic number $\alpha \in \overline{\mathbb{Q}}$ [Adleman et al. 1997], or the sum $\vec{t}_1 + \vec{t}_2$ of two superpositions. We represent the set of superpositions as S, and superpositions themselves are denoted by $\vec{r}$, $\vec{s}$, $\vec{t}_1$, $\vec{t}_2$, and so forth. In essence, terms correspond to objects that can reduce classically, possibly to a superposition, while superpositions represent quantum computations. We define the sets BV and V as the sets of *basis values* and *values*, respectively. Basis values are a subset of terms in normal form, and values are superpositions of basis values. We use v, w, v_1, v_2, and so on for basis values, while $\vec{v}$, $\vec{w}$, $\vec{v}_1$, $\vec{v}_2$, and so on, denote values.

A variable is free in a superposition if it is not bound by an abstraction or a pair destructor. We denote the set of free variables in the superposition $\vec{t}$ as $FV(\vec{t})$. A superposition is closed if it contains no free variables. Given a set $\mathcal{S}$ of superpositions, we define $\mathcal{S}_c$ as the set of closed superpositions within $\mathcal{S}$. A PUNQ program is a closed superposition in S_c that can be assigned a type according to the type discipline presented in Section 3.

The size of a superposition $\vec{t}$, denoted $|\vec{t}|$, is the maximal size of its superposed terms. Formally,

$$
\begin{array}{rlrl}
|x| &\triangleq 1 & |t_1\ t_2| = |(t_1,t_2)| &\triangleq |t_1| + |t_2| + 1 \\
||0\rangle| &\triangleq 1 & |\text{let } (x,y) = t \text{ in } \vec{t}| &\triangleq 1 + |t| + |\vec{t}| \\
||1\rangle| &\triangleq 1 & |\vec{0}| &\triangleq 0 \\
|\text{if } t \text{ then } \vec{t}_1 \text{ else } \vec{t}_2| &\triangleq 1 + |t| + \max(|\vec{t}_1|, |\vec{t}_2|) & |\alpha \cdot \vec{t}| &\triangleq |\vec{t}| \\
|\lambda x.\vec{t}| &\triangleq 1 + |\vec{t}| & |\vec{t}_1 + \vec{t}_2| &\triangleq \max(|\vec{t}_1|, |\vec{t}_2|)
\end{array}
$$

The set S of superpositions has the structure of a vector space and, consequently, we define an equivalence relation $\equiv$ on S as follows:

$$
\begin{array}{rlrl}
\vec{t}_1 + \vec{t}_2 &\equiv \vec{t}_2 + \vec{t}_1 & (\vec{t}_1 + \vec{t}_2) + \vec{t}_3 &\equiv \vec{t}_1 + (\vec{t}_2 + \vec{t}_3) \\
\vec{0} + \vec{t} &\equiv \vec{t} & 0 \cdot \vec{t} &\equiv \vec{0} \\
1 \cdot \vec{t} &\equiv \vec{t} & \alpha \cdot (\beta \cdot \vec{t}) &\equiv \alpha\beta \cdot \vec{t} \\
\alpha \cdot \vec{t} + \beta \cdot \vec{t} &\equiv (\alpha + \beta) \cdot \vec{t} & \alpha \cdot (\vec{t}_1 + \vec{t}_2) &\equiv \alpha \cdot \vec{t}_1 + \alpha \cdot \vec{t}_2
\end{array}
$$

This also implies that the summation symbol $\sum$ can be used unambiguously. A superposition $\vec{t}$ is in *canonical form* if it is either $\vec{0}$ or $\vec{t} = \sum_{i=1}^{n} \alpha_i \cdot t_i$, where $\forall i \neq j, t_i \neq t_j$, and $\forall i,\ \alpha_i \neq 0$. The canonical form of a superposition $\vec{t}$ is unique, modulo associativity and commutativity. We denote the set of canonical forms as CF. We also define the syntactic sugar given in Figure 3.

2.2 Operational Semantics

The semantics of PUNQ programs is defined by the rewrite relation $\rightsquigarrow \subseteq \mathsf{S}_c \times \mathsf{S}_c$, given in Figure 4. We denote its reflexive and transitive closure as $\rightsquigarrow^*$.

$$\text{if } \sum_{i=1}^{n} \alpha_i \cdot s_i \text{ then } \vec{r}_1 \text{ else } \vec{r}_2 \triangleq \sum_{i=1}^{n} \alpha_i \cdot \text{if } s_i \text{ then } \vec{r}_1 \text{ else } \vec{r}_2$$

$$s \left(\sum_{i=1}^{n} \alpha_i \cdot t_i \right) \triangleq \sum_{i=1}^{n} \alpha_i \cdot s\ t_j$$

$$\left(\sum_{i=1}^{n} \alpha_i \cdot s_i, \sum_{j=1}^{m} \beta_j \cdot t_j \right) \triangleq \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j \cdot \left(s_i, t_j\right)$$

$$\text{let } (x,y) = \sum_{i=1}^{n} \alpha_i \cdot s_i \text{ in } \vec{r} \triangleq \sum_{i=1}^{n} \alpha_i \cdot \left(\text{let } (x,y) = s_i \text{ in } \vec{r}\right)$$

Fig. 3. Syntactic sugar on PUNQ syntax.

$$\frac{}{\text{if } |0\rangle \text{ then } \vec{s} \text{ else } \vec{t} \rightsquigarrow \vec{s}}\ (\text{If}_0) \qquad \frac{}{\text{if } |1\rangle \text{ then } \vec{s} \text{ else } \vec{t} \rightsquigarrow \vec{t}}\ (\text{If}_1) \qquad \frac{}{(\lambda x.\vec{t})\ v \rightsquigarrow \vec{t}[v/x]}\ (\text{Abs})$$

$$\frac{}{\text{let } (x,y) = (v,w) \text{ in } \vec{t} \rightsquigarrow \vec{t}[v/x, w/y]}\ (\text{Let}) \qquad \frac{t \rightsquigarrow \vec{s}}{\text{if } t \text{ then } \vec{r}_1 \text{ else } \vec{r}_2 \rightsquigarrow \text{if } \vec{s} \text{ then } \vec{r}_1 \text{ else } \vec{r}_2}\ (\text{If}_+)$$

$$\frac{t \rightsquigarrow \vec{s}}{r\ t \rightsquigarrow r\ \vec{s}}\ (\text{App}) \qquad \frac{t \rightsquigarrow \vec{s}}{t\ v \rightsquigarrow \vec{s}\ v}\ (\text{App}_\mathsf{V}) \qquad \frac{t \rightsquigarrow \vec{s}}{(t,r) \rightsquigarrow (\vec{s},r)}\ (\text{Pair}) \qquad \frac{t \rightsquigarrow \vec{s}}{(v,t) \rightsquigarrow (v,\vec{s})}\ (\text{Pair}_\mathsf{V})$$

$$\frac{t \rightsquigarrow \vec{s}}{\text{let } (x,y) = t \text{ in } \vec{r} \rightsquigarrow \text{let } (x,y) = \vec{s} \text{ in } \vec{r}}\ (\text{Let}_+)$$

$$\frac{\sum_{i\in I} \alpha_i \cdot t_i + \sum_{j\in J} \beta_j \cdot v_j \in \text{CF} \qquad \forall i \in I,\ t_i \rightsquigarrow s_i}{\sum_{i\in I} \alpha_i \cdot t_i + \sum_{j\in J} \beta_j \cdot v_j \rightsquigarrow \sum_{i\in I} \alpha_i \cdot s_i + \sum_{j\in J} \beta_j \cdot v_j}\ (\text{Sup}) \qquad \frac{\vec{t} \equiv \vec{t}_1 \qquad \vec{t}_1 \rightsquigarrow \vec{s}_1 \qquad \vec{s}_1 \equiv \vec{s}}{\vec{t} \rightsquigarrow \vec{s}}\ (\text{Equ})$$

Fig. 4. Semantics of PUNQ programs.

It is important to emphasize that the syntactic sugar defined in Figure 3 is used in the reduction rules (If_+), (App), (App_V), (Pair), (Pair_V), and (Let_+) to simplify notations. For instance, the reduction of rule (If_+) can be written as

$$\frac{t \rightsquigarrow \sum_i \alpha_i \cdot s_i}{\text{if } t \text{ then } \vec{r}_1 \text{ else } \vec{r}_2 \rightsquigarrow \sum_i \alpha_i \cdot \text{if } s_i \text{ then } \vec{r}_1 \text{ else } \vec{r}_2}\ (\text{If}_+)$$

Therefore, reductions through $\rightsquigarrow$ do not generate terms or superpositions that are not syntactically valid. Another crucial point to note is that the relation $\rightsquigarrow$ is constrained to canonical forms in rule (Sup) to prevent the reduction of a superposition in the form $v + 0 \cdot t$, where $t \in \mathsf{T}$, $v \in \mathsf{V}$, as it is true that $v + 0 \cdot t \equiv v$ and, as a result, $v + 0 \cdot t$ should not reduce.

The semantics of PUNQ is call-by-value. Since there is an established strategy, the confluence of this calculus is trivial. The normal forms are closed values and they are unique modulo the equivalence relation $\equiv$.

3 A TYPE SYSTEM FOR UNITARITY AND POLYTIME NORMALIZATION

In this section, we introduce a type system ensuring that typable closed superpositions:

- encode unitary transformations on the type of circuits over qubits (i.e., linear functions from qubits to qubits) (Section 5);
- normalize in time polynomial in their size, exhibiting only polynomial growth on the size of the superposition (Section 6).

The typing discipline is created by mixing the unitary-ensuring type system of Lambda-$\mathcal{S}_1$ [Díaz-Caro and Mal 2022] together with the polytime strong normalization properties of the DLAL [Baillot and Terui 2009] type system.

3.1 Types, Judgments, and Environments

The set $\mathbb{T}$ of types in PUNQ is generated by the following grammars:

$$\text{(PUNQ types)} \qquad \mathbb{T} \ni A, B, C, \ldots := X \mid A \multimap A \mid A \Rightarrow A \mid A \times A \mid Q \mid \S A \mid \forall X.A$$

$$\text{(Ground types)} \qquad Q, R, S, \ldots := \mathbb{B} \mid \sharp Q \mid \S Q \mid Q \times Q$$

We use A, B, C, and so on for PUNQ types, and $Q, R, S, \ldots$ for ground types. Types include type variables X, a basic type $\mathbb{B}$ for bits, a linear arrow $\multimap$, an intuitionistic arrow $\Rightarrow$, a type construct $\times$ for pairs corresponding to the tensor product of linear logic (we do not use the tensor notation of linear logic to avoid confusion as pairs only correspond to separable states), a modality $\sharp$ for superpositions, a modality $\S$ as a marker for possible duplication, and polymorphism. The set of closed types is denoted as $\mathbb{T}_c$. Ground types Q represent all types that can be inhabited by tuples of qubits. Intuitively, objects of type $\sharp Q$ are unitary superpositions of elements of type Q and hence cannot be cloned (i.e., duplicated). For example, $\sharp\mathbb{B}$ is the type of a unitary superposition of bits, i.e., qubits. Qubits correspond to values whose canonical forms are of the shape $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$, with $\alpha, \beta \in \overline{\mathbb{Q}}$ and $|\alpha|^2 + |\beta|^2 = 1$. We write A^n, with $n \geq 1$, as a shorthand for $A \times \ldots \times A$, n times. The type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)$ corresponds to quantum circuits over n qubits. Finally, a type of the form $\sharp\mathbb{B} \Rightarrow A$, for any type A, will be disallowed by the typing discipline as $\Rightarrow$ is not linear and hence could involve duplication or deletion of the input.

A *typing environment* Γ is a mapping from variables to closed types in $\mathbb{T}_c$ and is sometimes written as $x_1 : A_1, \ldots, x_n : A_n$. The notation Γ, Δ represents the disjoint union of the typing environments Γ and Δ. We write as $FV(\Gamma)$ the set of free type variables appearing in Γ.

Typing judgments are of the form $\Gamma; \Delta \vdash \vec{t} : A$, where Γ and Δ are disjoint typing environments, $\vec{t}$ is a superposition, and A is a type. Here, Γ is referred to as the *exponential context*, and Δ is the *linear context*.

3.2 Orthogonality

We define the inner product $\langle -|-\rangle : \mathsf{V}_c \times \mathsf{V}_c \to \overline{\mathbb{Q}}$ over closed values as

$$\left\langle \sum_{i=1}^{n} \alpha_i \cdot v_i \;\middle|\; \sum_{j=1}^{m} \beta_j \cdot w_j \right\rangle \triangleq \sum_{i=1}^{n}\sum_{j=1}^{m} \overline{\alpha_i}\beta_j \delta_{v_i,w_j}, \qquad \langle \vec{v}|\vec{0}\rangle = \langle \vec{0}|\vec{v}\rangle \triangleq 0,$$

where $\delta_{x,y}$ is the Kronecker delta that is equal to 1 if $x = y$ and 0 otherwise. Notice that inner product is obviously preserved by the equivalence relation $\equiv$ on the vector space of values. Hence V_c is a Hilbert space as $\ell^2(\mathsf{V}_c) = \mathsf{V}_c$.

Type checking a superposition will require that we are able to test orthogonality between terms. This is straightforward to do for values in normal form, but we may also face the situation where

$$\frac{A' \leq A \quad B \leq B'}{A \multimap B \leq A' \multimap B'} \qquad \frac{A' \leq A \quad B \leq B'}{A \Rightarrow B \leq A' \Rightarrow B'} \qquad \frac{A \leq A' \quad B \leq B'}{A \times B \leq A' \times B'} \qquad \frac{A \leq B}{\S A \leq \S B} \qquad \frac{A \leq B}{\forall X.A \leq \forall X.B}$$

$$\frac{}{A \leq A} \qquad \frac{}{Q \leq \sharp Q} \qquad \frac{}{\sharp\sharp Q \leq \sharp Q} \qquad \frac{}{Q \leq \sharp!(Q)} \qquad \frac{}{\sharp\S Q \leq \S\sharp Q} \qquad \frac{A \leq B \quad B \leq C}{A \leq C}$$

Fig. 5. Subtyping relation.

a term correctly represents a superposition, and yet it is not fully reduced, such as the term $\frac{1}{\sqrt{2}} \cdot (\lambda x.x)|0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle$. Therefore, we use the following definition of orthogonality between terms.

Definition 3.1 (Orthogonality). The *orthogonality relation* $\perp \subseteq \mathsf{S}_c \times \mathsf{S}_c$ is defined as $\vec{t} \perp \vec{s}$ if and only if $\vec{t} \rightsquigarrow^* \vec{v}$, $\vec{s} \rightsquigarrow^* \vec{w}$ and $\langle \vec{v} | \vec{w} \rangle = 0$.

For a given typing environment Γ, let σ_Γ denote a type-preserving substitution of variables in Γ by closed basis values. The orthogonality relation can be extended to any two superpositions $\vec{t},\ \vec{s} \in \mathsf{S}$ such that $\Gamma; \Delta \vdash \vec{t} : A$ and $\Gamma; \Delta \vdash \vec{s} : A$ by defining $(\vec{t} \perp_A^{\Gamma,\Delta} \vec{s})$ iff $\forall \sigma_{\Gamma\cup\Delta},\ \vec{t}\sigma_{\Gamma\cup\Delta} \perp_A \vec{s}\sigma_{\Gamma\cup\Delta}$.

3.3 Type System

We introduce a *bang function* on types, which the type system utilizes to remove the $\sharp$ modalities when they correspond to a non-linear use.

Definition 3.2 (Bang function). The function $! : \mathbb{T} \to \mathbb{T}$ is an endomorphism on types defined as

$$!(X) \triangleq X \qquad !(\mathbb{B}) \triangleq \mathbb{B}$$
$$!(A \multimap B) \triangleq A \multimap B \qquad !(A \Rightarrow B) \triangleq A \Rightarrow B$$
$$!(A \times B) \triangleq !(A) \times !(B) \qquad !(\sharp Q) \triangleq !(Q)$$
$$!(\S A) \triangleq \S!(A) \qquad !(\forall X.A) \triangleq \forall X.!(A)$$

This function is reminiscent of the bang modality in linear logic [Girard 1987] because it transforms a non-clonable type (non-duplicable type, e.g., superposition) into a clonable type (duplicable type). Therefore, it corresponds to withdrawing $\sharp$ modalities. For instance, $!(\sharp\mathbb{B}) = \mathbb{B}$ represents the type of bits and is clonable. Similarly, $!(\sharp\mathbb{B} \multimap \sharp\mathbb{B}) = \sharp\mathbb{B} \multimap \sharp\mathbb{B}$ is the type of unitary maps on qubits (see Theorem 5.6), which is clonable by default.

The *subtyping* relation $\leq \subseteq \mathbb{T} \times \mathbb{T}$ is defined in Figure 5. The intuition behind this relation is as follows: if a type Q is considered as the base of a vector space for its values, then $\sharp Q$ results in the intersection of the span of A with the unitary sphere, that is, complex linear combinations of objects of type A with unit norm. Therefore, $\leq$ corresponds to set inclusion, and it holds that $Q \leq \sharp Q$ and $\sharp Q = \sharp\sharp Q$. Additionally, it can be shown that $\sharp Q \times \sharp R \leq \sharp(!(Q) \times !(R))$, implying the desirable property that the vector space of separable qubits $\sharp\mathbb{B} \times \sharp\mathbb{B}$ is included in the vector space of 2 qubits $\sharp(\mathbb{B} \times \mathbb{B})$.

Since the ! function will only be used in the double arrow in rule introduction $(\Rightarrow_i)$ on the left of that arrow, the double arrow carries all the information about whether the ! function was applied. Therefore, ! is not treated as a modality and appears implicitly in type substitution. Given a type C and a variable X, let $[C/X]$ represent the *type substitution* σ^+. It is inductively defined on types

(up to α-renaming) as follows:

$$\sigma^k(Y) \triangleq Y, \text{ with } Y \neq X, \qquad \sigma^k(A \Rightarrow B) \triangleq \sigma^-(A) \Rightarrow \sigma^+(B),$$
$$\sigma^+(X) \triangleq C, \qquad \sigma^k(\sharp Q) \triangleq \sharp\sigma^k(Q),$$
$$\sigma^-(X) \triangleq !(C), \qquad \sigma^k(\S A) \triangleq \S\sigma^k(A),$$
$$\sigma^k(\mathbb{B}) \triangleq \mathbb{B}, \qquad \sigma^k(A \times B) \triangleq \sigma^k(A) \times \sigma^k(B),$$
$$\sigma^k(A \multimap B) \triangleq \sigma^k(A) \multimap \sigma^k(B), \qquad \sigma^k(\forall Y.A) \triangleq \forall Y.\sigma^k(A),$$

where $k \in \{+,-\}$ and where ! is the bang function from Definition 3.2. This ensures that non-clonability is preserved on inputs to non-linear applications, e.g., $(X \Rightarrow X)[\sharp\mathbb{B}/X] = \sigma^-(X) \Rightarrow \sigma^+(X) = !(\sharp\mathbb{B}) \Rightarrow \sharp\mathbb{B} = \mathbb{B} \Rightarrow \sharp\mathbb{B}$.

The *typing rules* are provided in Figure 6. PUNQ is the set of typable closed superpositions in S_c.

Several key points are worth highlighting: (i) The ! function is employed in rule $(\Rightarrow_i)$ to ensure the no-cloning property. (ii) In rule $(\Rightarrow_e)$, the notation $[z : C]$ indicates that the variable z is optional: it can either appear in both the hypothesis and the conclusion of the rule or not at all. The variable z is then passed to the exponential context in the conclusion of the rule. Consequently, qubits are treated linearly following the DLAL type discipline, and they cannot be cloned by side effect. For instance, if z is of type $\sharp\mathbb{B}$, then $\lambda z.t\ s$ is of type $\mathbb{B} \Rightarrow B$, as per rule $(\Rightarrow_i)$. (iii) Rules $(\text{if}_\sharp)$ and $(\sharp_i)$ are the only two rules that make use of the orthogonality predicate $\perp_A \subseteq \mathbb{T}_c \times \mathbb{T}_c$. (iv) Rule $(\sharp_i)$ disallows superposing arrow types. (v) Rule $(\forall_e)$ utilizes the type substitution $[B/X]$.

PUNQ enjoys the following standards properties.

Theorem 3.3 (Subject reduction). *If* $;\vdash \vec{t} : A$ *and* $\vec{t} \rightsquigarrow \vec{r}$, *then* $;\vdash \vec{r} : A$.

Proof. By induction on the relation $\rightsquigarrow$. The full proof, including the needed substitution lemma, is given in Appendix A.1. □

Theorem 3.4 (Progress). *If* $;\vdash \vec{t} : A$, *then either* $\vec{t} \rightsquigarrow \vec{r}$ *for some* $\vec{r}$ *or* $\vec{t} \in \mathsf{V}$.

Proof. By induction on the structure of $\vec{t}$. Full proof given in Appendix A.1. □

3.4 Intuitions

Most rules in Figure 6 are reminiscent of the typing discipline of Dual Light Affine Logic [Baillot and Terui 2009], extended with multiplicative pairs (cf. Figure 10). The weakening rule (W) and contraction rule (C) are limited to exponential contexts since discarding or duplicating a qubit would break unitarity. In our setting, the typing rules are linear in the linear context.

Rule (≡) guarantees a type preservation property on the vector space of superpositions. This rule is crucial for subject reduction. Indeed, terms of a typable superposition must have a norm of 1 (by rule $(\sharp_i)$), preventing a superposition of the form $\frac{1}{2} \cdot t + \frac{1}{2} \cdot t$ from typing, even though it may be obtained as a reduct of the operational semantics. An alternative approach could have been restricting the relation $\rightsquigarrow$ to pairs of canonical forms in $\mathsf{S}_c/_\equiv \times \mathsf{S}_c/_\equiv$ and evaluating arbitrary superpositions only using their canonical representative.

Rule $(\leq)$ is the subtyping rule, allowing the system to handle entangled datatypes.

The typing rules (if) and $(\text{if}_\sharp)$ are additive rules for conditionals on type A controlled by classical data $\mathbb{B}$ and quantum data $\sharp\mathbb{B}$, respectively. In this latter case, the conditional evaluates to a superposition of conditionals, as per rule (If_+) in Figure 4. Hence, the result has type $\sharp A$.

The typing rules for linear and intuitionistic arrows resemble those in [Barber and Plotkin 1996], with the additional requirement that the introduction and elimination of the intuitionistic arrow,

$$\frac{\Gamma;\Delta \vdash \vec{t} : A}{\Gamma,\Gamma';\Delta \vdash \vec{t} : A}\ (\mathsf{W}) \qquad \frac{\Gamma, x : B, y : B;\Delta \vdash \vec{t} : A}{\Gamma, x : B;\Delta \vdash \vec{t}[x/y] : A}\ (\mathsf{C}) \qquad \frac{\Gamma;\Delta \vdash \vec{t} : A \quad \vec{t} \equiv \vec{s}}{\Gamma;\Delta \vdash \vec{s} : A}\ (\equiv)$$

$$\frac{\Gamma;\Delta \vdash \vec{t} : A \quad A \leq B}{\Gamma;\Delta \vdash \vec{t} : B}\ (\leq) \qquad \frac{}{;x : A \vdash x : A}\ (\mathsf{Ax}) \qquad \frac{}{;\vdash |0\rangle : \mathbb{B}}\ (0) \qquad \frac{}{;\vdash |1\rangle : \mathbb{B}}\ (1)$$

$$\frac{\Gamma;\Delta \vdash t : \sharp\mathbb{B} \quad \Gamma';\Delta';\vdash \vec{s}_1 : Q \quad \Gamma';\Delta' \vdash \vec{s}_2 : Q \quad \vec{s}_1 \perp^{\Gamma';\Delta'} \vec{s}_2}{\Gamma,\Gamma';\Delta,\Delta' \vdash \mathsf{if}\ t\ \mathsf{then}\ \vec{s}_1\ \mathsf{else}\ \vec{s}_2 : \sharp Q}\ (\mathsf{if}_\sharp)$$

$$\frac{;\Gamma,\Delta \vdash \vec{t} : A}{\Gamma;\S\Delta \vdash \vec{t} : \S A}\ (\S_i) \qquad \frac{\Gamma;\Delta \vdash t : \mathbb{B} \quad \Gamma';\Delta' \vdash \vec{s_1} : A \quad \Gamma';\Delta' \vdash \vec{s_2} : A}{\Gamma,\Gamma';\Delta \vdash \mathsf{if}\ t\ \mathsf{then}\ \vec{s}_1\ \mathsf{else}\ \vec{s}_2 : A}\ (\mathsf{if})$$

$$\frac{\Gamma;\Delta \vdash s : \S B \quad \Gamma';\Delta', x : \S B \vdash t : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t[s/x] : A}\ (\S_e) \qquad \frac{\Gamma;\Delta \vdash \vec{t} : A \quad X \notin FV(\Gamma,\Delta)}{\Gamma;\Delta \vdash \vec{t} : \forall X.A}\ (\forall_i)$$

$$\frac{\forall i,\ \Gamma;\Delta \vdash \vec{t}_i : Q \quad \forall j \neq k,\ \vec{t}_j \perp^{\Gamma;\Delta} \vec{t}_k \quad \sum_{i=1}^{n} |\alpha_i|^2 = 1}{\Gamma;\Delta \vdash \sum_{i=1}^{n} \alpha_i \cdot \vec{t}_i : \sharp Q}\ (\sharp_i) \qquad \frac{\Gamma;\Delta \vdash \vec{t} : \forall X.A}{\Gamma;\Delta \vdash \vec{t} : A[B/X]}\ (\forall_e)$$

$$\frac{\Gamma;\Delta, x : A \vdash \vec{t} : B}{\Gamma;\Delta \vdash \lambda x.\vec{t} : A \multimap B}\ (\multimap_i) \qquad \frac{\Gamma;\Delta \vdash t : A \multimap B \quad \Gamma';\Delta' \vdash s : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t\ s : B}\ (\multimap_e)$$

$$\frac{\Gamma, x : A;\Delta \vdash \vec{t} : B}{\Gamma;\Delta \vdash \lambda x.\vec{t} : !(A) \Rightarrow B}\ (\Rightarrow_i) \qquad \frac{\Gamma;\Delta \vdash t : A \Rightarrow B \quad ;[z : C] \vdash s : A}{\Gamma,[z : C];\Delta \vdash t\ s : B}\ (\Rightarrow_e)$$

$$\frac{\Gamma;\Delta \vdash t : A \quad \Gamma';\Delta' \vdash s : B}{\Gamma,\Gamma';\Delta,\Delta' \vdash (t,s) : A \times B}\ (\times_i) \qquad \frac{\Gamma;\Delta \vdash t : A \times B \quad \Gamma';\Delta', x : A, y : B \vdash \vec{s} : C}{\Gamma,\Gamma';\Delta,\Delta' \vdash \mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s} : C}\ (\times_e)$$

$$\frac{\Gamma;\Delta \vdash t : \sharp(Q \times R) \quad \Gamma';\Delta', x : \sharp Q, y : \sharp R \vdash \vec{s} : S}{\Gamma,\Gamma';\Delta,\Delta' \vdash \mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s} : \sharp S}\ (\times_{e\sharp})$$

Fig. 6. Typing rules for PUNQ programs.

rules $(\Rightarrow_i)$ and $(\Rightarrow_e)$, can only be performed on "banged" data. The bang function ensures that the resulting type is copyable (or clonable) by turning a qubit into a bit, $!(\sharp\mathbb{B}) = \mathbb{B}$, propagating over pairs, and not changing applications. For instance, $!(\sharp\mathbb{B} \multimap \sharp\mathbb{B}) = \sharp\mathbb{B} \multimap \sharp\mathbb{B}$, as it is safe to copy a quantum gate. Therefore, the intuitionistic arrow explicitly forbids values as inputs, and types of the form $\sharp A \Rightarrow B$ are uninhabited.

The pair constructor and separable pair destructor are typed using the multiplicative rules $(\times_i)$ and $(\times_e)$, following the encoding of [Baillot and Terui 2009]. Rule $(\times_{e\sharp})$ is the rule for the quantum pair destructor, allowing the handling of a superposition of pairs, hence a possibly entangled state. In this setting, the pair can be viewed as the tensor product used in quantum computing. Variables x and y are typed by $\sharp Q$ and $\sharp R$ to preserve unitarity. Note that this typing rule can be extended

to superpositions as follows:

$$\frac{\Gamma_1;\Delta_1 \vdash \sum_i \alpha_i \cdot t_i : \sharp(Q \times R) \qquad \Gamma_2;\Delta_2, x : \sharp Q, y : \sharp R \vdash \vec{s} : S}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \sum_i \alpha_i \cdot \mathsf{let}\ (x,y) = t_i\ \mathsf{in}\ \vec{s} : \sharp S}\ (\vec{\times}_{e\sharp})$$

Rule $(\vec{\times}_{e\sharp})$ can be simulated using rules $(\times_{e\sharp})$ and $(\sharp_i)$ but has the advantage of providing a more flexible typing discipline and simplifying the orthogonality requirements of rule $(\sharp_i)$. Indeed, orthogonality is tested on subterms t_i rather than on terms $\mathsf{let}\ (x,y) = t_i\ \mathsf{in}\ \vec{s}$.

DLAL characterizes the complexity class FPTIME [Baillot and Terui 2009] by imposing constraints on the use of the contraction rule in proofs and introducing a new modality § in rules $(\S_i)$ and $(\S_e)$ to address some of these limitations. Broadly speaking, the modality § serves as a marker for objects resulting from iterations that cannot be further iterated. Using this construct limits the number of iterations possible in proofs and, consequently, the complexity of the system. Specifically, DLAL enforces a design principle known as *stratification* by restricting the introduction and elimination of the modality §. This stratification is not sufficient on its own to characterize polynomial time (it provides a characterization of elementary time [Girard 1998]) and this is the reason why rule $(\Rightarrow_e)$ is restricted to at most one open variable.

Rule $(\sharp_i)$ is the introduction rule for superpositions. A superposition of objects can be created under the conditions that they share the same type, are pairwise orthogonal according to the predicate $\perp_A$, that the norm of the generated superposition $\sum_{i=1}^{n} |\alpha_i|^2$ equals 1, and that the type A is not an arrow type. Implicit in this rule is the fact that the construction of superpositions is limited to complex amplitudes α_i that are polytime approximable, a set which we denote by $\overline{\mathbb{Q}}$ throughout the paper. When considering quantum computations restricted to some polytime complexity class, it is standard to include such restrictions since, for instance, intractable problems could be encoded into transition amplitudes [Adleman et al. 1997; Bernstein and Vazirani 1997].

Our notion of orthogonality is that of [Díaz-Caro and Malherbe 2022], where the superposition of functions is disallowed. It is treated more realistically than in [Díaz-Caro et al. 2019b], where the superposition of functions is allowed, but it is generally not a function.

4 EXAMPLES

We may now consider some examples of typing judgements in PUNQ which correspond to different tools in quantum computation. Informally, a term t is said to represent a unitary operator if, when applied to another term encoding some superposition, the application reduces to what would be the PUNQ representation of the output of the operator. A precise definition of *representing a quantum gate or a unitary operator* is given in Section 5.2.

4.1 Superpositions

To demonstrate the typing of a superposition we will consider the example of the X-basis states $|\pm\rangle \triangleq \frac{1}{\sqrt{2}} \cdot |0\rangle \pm \frac{1}{\sqrt{2}} \cdot |1\rangle$, also called the *plus* and *minus* states, which correspond to the unit-norm eigenstates of the Pauli-X operator.

The conditions in the typing rule $(\sharp_i)$ correspond to checking that the basis terms have the correct type (i.e. since we are interested in typing a qubit state, these terms should be either booleans or qubits), that they are mutually orthogonal, and that the resulting state has unit norm.

Example 4.1 (X-basis states). The one-qubit states $|\pm\rangle$ have type $\sharp\mathbb{B}$.

$$\pi_{\pm} \triangleq \frac{\dfrac{}{;\vdash |0\rangle : \mathbb{B}}\ (0) \qquad \dfrac{}{;\vdash |1\rangle : \mathbb{B}}\ (1) \qquad |0\rangle \perp |1\rangle \qquad \left|\tfrac{1}{\sqrt{2}}\right|^2 + \left|\pm\tfrac{1}{\sqrt{2}}\right|^2 = 1}{;\vdash \frac{1}{\sqrt{2}} \cdot |0\rangle \pm \frac{1}{\sqrt{2}} \cdot |1\rangle : \sharp\mathbb{B}}\ (\sharp_i)$$

Notice that, if we use qubit terms as the basis to form the superposition (i.e. values of type $\sharp\mathbb{B}$) then their superposition will have type $\sharp\sharp\mathbb{B}$ by rule $(\sharp_i)$ but this corresponds to the typical qubit type $\sharp\mathbb{B}$ and can be enforced via subtyping since $\sharp\sharp Q \leq \sharp Q$.

4.2 Single-Qubit Gates

We will now turn to the construction of single-qubit gates which will form the basic building blocks for expressivity in the PUNQ language.

We will consider the specific examples of the Hadamard and Z gates but we will see that their construction technique can be applied for any single-qubit gate. Remember that a necessary and sufficient condition of any unitary matrix $U : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$ is that its columns form an orthogonal basis of $\mathbb{C}^{2^n}$, meaning that they are pairwise orthogonal and have unit norm. These are precisely the requirements of the quantum control typing rule $(\text{if}_\sharp)$.

Example 4.2 (Hadamard). Let $\pi_\pm$ represent the derivation trees for $|+\rangle$ and $|-\rangle$ in Example 4.1. The term $\mathsf{H} \triangleq \lambda x.\text{if } x \text{ then } |+\rangle \text{ else } |-\rangle$ representing the Hadamard gate has type $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$:

$$\dfrac{\dfrac{\dfrac{}{;x:\sharp\mathbb{B} \vdash x:\sharp\mathbb{B}}\,(\text{Ax}) \qquad \begin{matrix}\pi_+ \\ \vdots \\ ;\vdash |+\rangle:\sharp\mathbb{B}\end{matrix} \qquad \begin{matrix}\pi_- \\ \vdots \\ ;\vdash |-\rangle:\sharp\mathbb{B}\end{matrix} \qquad |+\rangle \perp |-\rangle}{;x:\sharp\mathbb{B} \vdash \text{if } x \text{ then } |+\rangle \text{ else } |-\rangle : \sharp\sharp\mathbb{B}}\,(\text{if}_\sharp)}{;x:\sharp\mathbb{B} \vdash \text{if } x \text{ then } |+\rangle \text{ else } |-\rangle : \sharp\mathbb{B}}\,(\leq)}{;\vdash \mathsf{H} : \sharp\mathbb{B} \multimap \sharp\mathbb{B}}\,(\multimap_i)$$

Example 4.3 (Z gate). The term $\mathsf{Z} \triangleq \lambda x.\text{if } x \text{ then } |0\rangle \text{ else } -1 \cdot |1\rangle$ representing the Pauli Z gate can also be typed:

$$\dfrac{\dfrac{\dfrac{}{;x:\sharp\mathbb{B} \vdash x:\sharp\mathbb{B}}\,(\text{Ax}) \qquad \dfrac{\dfrac{}{;\vdash |0\rangle:\mathbb{B}}\,(0)}{;\vdash |0\rangle:\sharp\mathbb{B}}\,(\leq) \qquad \dfrac{\dfrac{}{;\vdash |1\rangle:\mathbb{B}}\,(1)}{;\vdash -1\cdot|1\rangle:\sharp\mathbb{B}}\,(\sharp_i) \qquad |0\rangle \perp |1\rangle}{;x:\sharp\mathbb{B} \vdash \text{if } x \text{ then } |0\rangle \text{ else } -1\cdot|1\rangle : \sharp\sharp\mathbb{B}}\,(\text{if}_\sharp)}{;x:\sharp\mathbb{B} \vdash \text{if } x \text{ then } |0\rangle \text{ else } -1\cdot|1\rangle : \sharp\mathbb{B}}\,(\leq)}{;\vdash \mathsf{Z} : \sharp\mathbb{B} \multimap \sharp\mathbb{B}}\,(\multimap_i)$$

Using nested if statements, it is possible to construct larger gates. In fact, it is not hard to show that this suffices to construct *all* unitary gates (see proof of Theorem 5.7). However, it is much more common to reason about complex operations not in their matrix representation but rather as compositions of small gates. Therefore, PUNQ allows for the typing of larger gates defined from smaller ones.

4.3 Quantum Controlled Gates

The quantum controlled-*NOT* gate is a unitary operation on two qubits which performs a bitswitch on the second qubit depending on the state of the first, i.e. it is the linear operator that for each basis state $|xy\rangle$ returns $|x(y \oplus x)\rangle$, where $x, y \in \{0, 1\}$. The following example shows that, once we have successfully typed a term representing a unitary gate, creating its quantum controlled version in PUNQ is straightforward.

Example 4.4 (Controlled-NOT gate). Let $\mathsf{NOT} \triangleq \lambda x.\text{if } x \text{ then } |1\rangle \text{ else } |0\rangle$ which can be derived with type $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$. We can derive the following term with type $\sharp(\mathbb{B} \times \mathbb{B}) \multimap \sharp(\mathbb{B} \times \mathbb{B})$:

$$\mathsf{CNOT} \triangleq \lambda z.\text{let } (x, y) = z \text{ in if } x \text{ then } (|0\rangle, y) \text{ else } (|1\rangle, \mathsf{NOT}\ y)$$

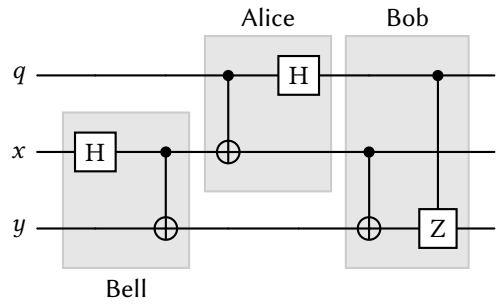


Fig. 7. Circuit for quantum teleportation with delayed measurements.

$$\dfrac{\dfrac{}{;z:\sharp(\mathbb{B}^2)\vdash z:\sharp(\mathbb{B}^2)}\ (\text{Ax}) \qquad \dfrac{\dfrac{\dfrac{}{;x:\sharp\mathbb{B}\vdash x:\sharp\mathbb{B}}\ (\text{Ax}) \qquad \dfrac{\vdots}{;y:\sharp\mathbb{B}\vdash \text{NOT}\ y:\sharp\mathbb{B}}}{;y:\sharp\mathbb{B}\vdash(|0\rangle,y)\perp(|1\rangle,\text{NOT}\ y):\sharp(\mathbb{B}^2)}\ (\text{if}_\sharp)}{;x,y:\sharp\mathbb{B}\vdash \text{if}\ x\ \text{then}\ (|0\rangle,y)\ \text{else}\ (|1\rangle,\text{NOT}\ y):\sharp\sharp(\mathbb{B}^2)}\ (\leq)}{;x,y:\sharp\mathbb{B}\vdash \text{if}\ x\ \text{then}\ (|0\rangle,y)\ \text{else}\ (|1\rangle,\text{NOT}\ y):\sharp(\mathbb{B}^2)}\ (\times_{e\sharp})$$

$$\dfrac{;z:\sharp(\mathbb{B}^2)\vdash \text{let}\ (x,y)=z\ \text{in if}\ x\ \text{then}\ (|0\rangle,y)\ \text{else}\ (|1\rangle,\text{NOT}\ y):\sharp(\mathbb{B}^2)}{;\vdash \text{CNOT}:\sharp(\mathbb{B}^2)\multimap\sharp(\mathbb{B}^2)}\ (\multimap_i)$$

Notice that the derivation does not require any particular properties of the *NOT* gate, only that the term NOT representing it has type $\sharp\mathbb{B}\multimap\sharp\mathbb{B}$. The choice of using the values $|0\rangle$ and $|1\rangle$ to encode the value of x in each branch of CNOT comes from the fact that the PUNQ typing system prohibits direct reuse of the variable x since, in general, this can break unitarity. Hence, reusing a qubit variable inside any branch controlled on it is not permitted.

4.4 Composition

One may also use PUNQ to define larger operations operations from smaller ones by composing them. To demonstrate this, we will consider the example of quantum teleportation [Bennett et al. 1993] with delayed measurements, and show that by defining smaller terms it is easy to describe a term that represents the total operation.

Example 4.5 (Quantum teleportation). Using terms H and CNOT as defined in Examples 4.2 and 4.4, we may define a term representing the linear map $|ab\rangle\mapsto\frac{1}{\sqrt{2}}\left(|0b\rangle+(-1)^a\cdot|1(1-b)\rangle\right)$:

$$\text{Bell}\triangleq\lambda z.\text{let}\ (x,y)=z\ \text{in}\ \text{CNOT}\,(\text{H}\,x,y):\sharp(\mathbb{B}^2)\multimap\sharp(\mathbb{B}^2)$$

In the specific case where $a=b=0$, it produces the Bell state $\frac{1}{\sqrt{2}}\big(|00\rangle+|11\rangle\big)$, which is used as a resource in teleportation. We can also encode Alice's and Bob's actions:

$$\begin{aligned}\text{Alice}&\triangleq\lambda a.\text{let}\ (q,x)=\text{CNOT}\ a\ \text{in}\ (\text{H}\,q,x)\\ \text{Bob}&\triangleq\lambda b.\text{let}\ (w,y)=b\ \text{in}\\ &\qquad\text{let}\ (q,x)=w\ \text{in}\ (\text{if}\ q\ \text{then}\ (|0\rangle,\text{CNOT}\,(x,y))\ \text{else}\ (|1\rangle,\text{CNOT}\,(x,\text{Z}\,y)))\end{aligned}$$

with types $\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2)$ and $\sharp(\mathbb{B}^3) \multimap \sharp(\mathbb{B}^3)$, respectively. The final term can be obtained, representing the full operation as depicted in Figure 7:

$$\mathsf{telep} \triangleq \lambda z.\mathsf{let}\ (q,w) = z\ \mathsf{in}\ (\mathsf{let}\ (x,y) = \mathsf{Bell}\ w\ \mathsf{in}\ \mathsf{Bob}\ (\mathsf{Alice}\ (q,x)\,,y)) : \sharp(\mathbb{B}^3) \multimap \sharp(\mathbb{B}^3).$$

4.5 Iteration

So far we have only considered examples of constant-time operations. We will now turn to iteration in PUNQ which is inherited from the DLAL typing system.

Gate iteration stands at the intersection of the DLAL and the Lambda-$\mathcal{S}_1$ typing systems and is an example of the versatility that PUNQ can provide in safely mixing the two systems. As a toy example, consider the quantum random walk [Childs et al. 2003; Kempe 2003] as an illustrating example for polytime iteration. This technique is known to provide a quantum speedup over its classical analog in different applications [Ambainis 2003] and an illustrative circuit is provided in Figure 8b.

In PUNQ, as in DLAL, we can implement Church numerals with the following type and syntax:

$$\mathbb{N} \triangleq \forall X.(X \multimap X) \Rightarrow \S(X \multimap X) \qquad \underline{n} \triangleq \lambda f.\lambda x.f(f(\dots(f\ x)\dots) \quad (n \text{ times}).$$

For instance, we have that $\underline{2} \triangleq \lambda f.\lambda x.f(f\ x)$, which can be derived with type $\mathbb{N}$. DLAL is complete for polynomial time, and indeed, for any polynomial $P(n) = n^{2^d}$, there is a DLAL term poly with type $\mathbb{N} \multimap \S^{2d}\,\mathbb{N}$ that simulates it [Baillot and Terui 2009, Proposition 11], which can also be derived in PUNQ. We will now use the Church encoding from DLAL to perform gate iteration in the example of a quantum random walk.

Example 4.6 (Quantum random walk). We define the R and L operators, for some set of K nodes, correspond to "taking a step" in one direction or another in a given structure. For instance, if we consider a closed loop, we consider the unitary operators corresponding to a "right" step and a "left" step (see Figure 8a):

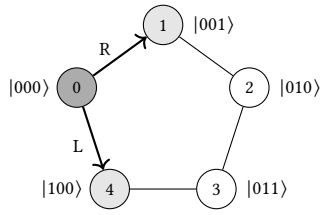
$$\mathsf{R}\,|i\rangle \triangleq |i+1 \pmod K\rangle \quad \text{and} \quad \mathsf{L}\,|i\rangle \triangleq |i-1 \pmod K\rangle, \quad i \in \{0,\dots,K-1\},$$

where the basis state $|i\rangle$ corresponds to the particle in node i. For $k = \lceil \log(K-1) \rceil$, the operators R and L correspond to unitary operators in the space $\overline{\mathbb{Q}}^{2^k} \to \overline{\mathbb{Q}}^{2^k}$, and therefore by Theorem 5.7 we show there exist corresponding PUNQ terms, say R and L, of type $\sharp(\mathbb{B}^k) \multimap \sharp(\mathbb{B}^k)$ that simulate them. A single step of the quantum walk can then be done with the following term:

$$\mathsf{step} \triangleq \lambda z.\mathsf{let}\ (x,y) = z\ \mathsf{in}\ (\mathsf{if}\ \mathsf{H}\ x\ \mathsf{then}\ (|0\rangle, \mathsf{L}\,y)\ \mathsf{else}\ (|1\rangle, \mathsf{R}\,y)) : \sharp(\mathbb{B}^{k+1}) \multimap \sharp(\mathbb{B}^{k+1})$$

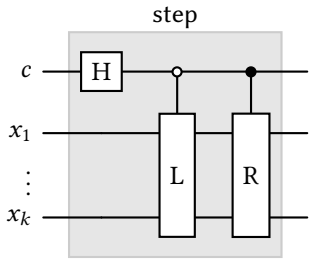
where x corresponds to the *coin* qubit deciding the direction of the step, on which we continuously apply the Hadamard gate. Since these terms correspond to an endomorphism, we may use the Church encoding of numerals to perform an iteration of step. For instance, this allows us to define an abstraction that takes in an integer input and iterates step precisely that number of times. Let $\mathsf{walk} \triangleq \lambda m.\lambda x.(m\ \mathsf{step})\ x$, which has type $\mathbb{N} \multimap \S\sharp(\mathbb{B}^{k+1}) \multimap \S\sharp(\mathbb{B}^{k+1})$ with the following derivation, where $A = \sharp(\mathbb{B}^{k+1})$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{;x:A,F:A\multimap A\vdash F\,x:A}}{;x:\S A,F:\S(A\multimap A)\vdash F\,x:\S A}\,(\S_i) \qquad \dfrac{\dfrac{\dfrac{}{;m:\mathbb{N}\vdash m:\mathbb{N}}\,(\mathsf{Ax})}{;m:\mathbb{N}\vdash m:(A\multimap A)\Rightarrow\S(A\multimap A)}\,(\forall_e) \qquad ;\vdash \mathsf{step}:A\multimap A}{;m:\mathbb{N}\vdash m\ \mathsf{step}:\S(A\multimap A)}\,(\Rightarrow_e)}{;m:\mathbb{N},x:\S A\vdash (m\ \mathsf{step})\ x:\S A}\,(\mathsf{C})}{;m:\mathbb{N}\vdash \lambda x.(m\ \mathsf{step})\ x:\S A\multimap \S A}\,(\multimap_i)}{;\vdash \mathsf{walk}:\mathbb{N}\multimap \S A\multimap \S A}\,(\multimap_i)$$



$$|0\rangle \otimes |000\rangle \rightarrow^{\text{step}} \tfrac{1}{\sqrt{2}}|0\rangle \otimes \mathrm{L}\,|000\rangle + \tfrac{1}{\sqrt{2}}|1\rangle \otimes \mathrm{R}|000\rangle$$

(a) A step in a loop with $K = 5$ nodes.



(b) Quantum circuit representation of step.

Fig. 8. Polytime quantum random walk.

For any $\underline{n}$ of type $\mathbb{N}$ we have that walk $\underline{n} \rightsquigarrow^* \lambda x.(\text{step}(\text{step } \ldots (\text{step } x)\ldots))$ where step is applied n times. Similarly, using the term poly encoding a polynomial P, we can construct an abstraction taking in an integer n that iterates step a number $P(n)$ of times:

$$\text{pwalk} \triangleq \lambda n.\lambda x.((\text{poly } n)\ \text{step})\ x : \mathbb{N} \multimap \S^{2d+1}\sharp(\mathbb{B}^{k+1}) \multimap \S^{2d+1}\sharp(\mathbb{B}^{k+1}).$$

4.6 No-Cloning and No-Deleting

In order to allow for cloning of variable while ensuring general good behavior for qubits, PUNQ incorporates different features of the DLAL and Lambda-$\mathcal{S}_1$ typing systems.

While DLAL exhibits full weakening (see Figure 10), there is a separation in the context between variables that may appear at most once (linear context) and more than once (exponential context), which then gets carried over into different types of abstractions. In Lambda-$\mathcal{S}_1$, on the other hand, we find a single context for typing judgements, and weakening is only allowed on so-called "flat types", where a type is restricted in its use of superpositions.

In PUNQ, the difference between the linear and exponential contexts captures the difference between allowed and disallowed treatment of qubits (or superpositions more generally), which then get resolved when creating the abstraction. Consider the following simple example of how PUNQ sensibly allows qubits to be the output of cloning, but not the input.

Example 4.7 (Qubits made from cloning). With two instances of rule (Ax) and one use of $(\times_i)$, it is straightforward to obtain the typing judgement $; x : \sharp\mathbb{B}, y : \sharp\mathbb{B} \vdash (x, y) : \sharp\mathbb{B} \times \sharp\mathbb{B}$. Now, moving x and y into the exponential context using rule $(\S_i)$ we obtain $x : \sharp\mathbb{B}, y : \sharp\mathbb{B}; \vdash (x, y) : \S(\sharp\mathbb{B} \times \sharp\mathbb{B})$ where we are now allowed to contract x and y using rule (C) and thus obtain $x : \sharp\mathbb{B}; \vdash (x, x) : \S(\sharp\mathbb{B} \times \sharp\mathbb{B})$.

Though it seems as if the validity of this typing judgement allows for cloning of the x variable representing a qubit, when we take the final step in closing the term, we must use rule $(\Rightarrow_i)$, since x appears in the exponential context. Combined with the fact that $!(\sharp\mathbb{B}) = \mathbb{B}$, we obtain $; \vdash \lambda x.\,(x, x) : \mathbb{B} \Rightarrow \S(\sharp\mathbb{B} \times \sharp\mathbb{B})$. Notice that this abstraction does not represent qubit cloning, but rather an operator that creates a pair of qubits using a boolean value that it takes as input to choose in which state the qubits should be.

In Section 5 we will show that PUNQ preserves unitarity over qubit maps in the same way as Lambda-$\mathcal{S}_1$ (Theorem 5.6), where in cases in which the number of qubits increases, the PUNQ term must behave as an isometry. Likewise, no term exists that represents a map that decreases the number of qubits, which would constitute qubit deletion. A consequence of this fact is that we may not reset qubits to some pre-fixed value, as shown in the following example.

Example 4.8 (Constant output zero). To create a lambda abstraction in which the input qubit does not appear in the expression, the variable must be introduced via weakening, which is limited to the exponential context. For instance, using rules (0), (≤), and (W) we may obtain the typing judgement $x : \sharp\mathbb{B}; \vdash |0\rangle : \sharp\mathbb{B}$. Applying rule $(\Rightarrow_i)$ to create the abstraction, we obtain $; \vdash \lambda x.|0\rangle : \mathbb{B} \Rightarrow \sharp\mathbb{B}$, which corresponds to "forgetting" a boolean input and giving as output a qubit in state zero.

5 UNITARITY

In PUNQ, we have adopted the Lambda-$\mathcal{S}_1$ typing discipline to ensure that linear maps over qubits, i.e., terms of type $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$, encode unitary operators. The proof of unitarity in [Díaz-Caro and Malherbe 2022] employs the realizability techniques developed in [Díaz-Caro et al. 2019b], which we will tailor to our setting. The major differences include a straightforward extension to tuples of qubits – demonstrating unitarity for types $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)$ – and the incorporation of DLAL constructs: the modality §, the intuitionistic arrow ⇒, as well as polymorphism. All the proofs omitted in this section are fully developed in Appendix A.2.

5.1 Realizability

The *base* of a closed value $\vec{v} \in \mathsf{V}_c$ is defined as the set of its basis values in BV_c:

$$\mathrm{base}(v) \triangleq \{v\} \qquad \mathrm{base}(\alpha \cdot \vec{v}) \triangleq \mathrm{base}(\vec{v}),\ \text{if } \alpha \neq 0,\ \emptyset \text{ otherwise.}$$
$$\mathrm{base}(\vec{0}) \triangleq \emptyset \qquad \mathrm{base}(\vec{v_1} + \vec{v_2}) \triangleq \mathrm{base}(\vec{v_1}) \cup \mathrm{base}(\vec{v_2})$$

Given a set S of closed values in a vector space V_c, the *span* (span) and *base* (♭) of S are defined as:

$$\mathrm{span}(S) \triangleq \Big\{ \sum_{i=1}^{n} \alpha_i \cdot \vec{v}_i : n \geq 0, \alpha_1, \ldots,\ \alpha_n \in \overline{\mathbb{Q}},\ \vec{v}_1, \ldots, \vec{v_n} \in S \Big\} \qquad \subseteq \mathsf{V}_c$$
$$\flat S \triangleq \bigcup_{\vec{v} \in S} \mathrm{base}(\vec{v}) \qquad \subseteq \mathsf{BV}_c$$

We first define the *realizability predicate* for a superposition $\vec{t}$ and set of closed values $S \subseteq \mathsf{V}_c$, written $\vec{t} \Vdash S$ that is true if and only if $\vec{t} \rightsquigarrow^* \vec{v}$ for some $\vec{v} \in S$. The *set of realizers of* S is then defined as $\{\vec{t} \in \mathsf{S}_c : \vec{t} \Vdash S\}$.

Let $\mathcal{S}_1 \subseteq \mathsf{V}_c$ be the set of unit values, that is, $\mathcal{S}_1 \triangleq \{\vec{v} \in \mathsf{V}_c : |\langle \vec{v} | \vec{v} \rangle|^2 = 1\}$. Given a function τ from type variables to subsets of $\mathcal{S}_1$, the *unitary semantics* $[\![\cdot]\!]_\tau : \mathbb{T} \to \mathcal{P}(\mathcal{S}_1)$ is defined in Figure 9 by induction on types.

$$[\![X]\!]_\tau \triangleq \tau(X), \qquad [\![A \multimap B]\!]_\tau \triangleq \big\{\lambda x.\vec{t} : \forall \vec{v} \in [\![A]\!]_\tau, (\lambda x.\vec{t})\vec{v} \Vdash [\![B]\!]_\tau\big\},$$
$$[\![\mathbb{B}]\!]_\tau \triangleq \{|0\rangle, |1\rangle\}, \qquad [\![A \Rightarrow B]\!]_\tau \triangleq \big\{\lambda x.\vec{t} : \forall v \in \flat[\![A]\!]_\tau, (\lambda x.\vec{t})v \Vdash [\![B]\!]_\tau\big\},$$
$$[\![\sharp A]\!]_\tau \triangleq \mathrm{span}([\![A]\!]_\tau) \cap \mathcal{S}_1, \qquad [\![A \times B]\!]_\tau \triangleq \big\{(\vec{v}, \vec{w}) : \vec{v} \in [\![A]\!]_\tau, \vec{w} \in [\![B]\!]_\tau\big\},$$
$$[\![\S A]\!]_\tau \triangleq [\![A]\!]_\tau, \qquad [\![\forall X.A]\!]_\tau \triangleq \bigcap_{R \subseteq \mathcal{S}_1} [\![A]\!]_{\tau \cup \{X \to R\}}.$$

Fig. 9. Unitary semantics $[\![\cdot]\!]_\tau : \mathbb{T} \to \mathcal{P}(\mathcal{S}_1)$.

The following property of the definition of ! (Definition 3.2) clarifies the nature of ♭ and the role of ! in erasing superposition.

Lemma 5.1. *Let $A \in \mathbb{T}_s$, then $[\![!(A)]\!]_\emptyset = \flat[\![A]\!]_\emptyset$.*

Proof. By induction on $A \in \mathbb{T}_s$. The full proof is given in Appendix A.2. □

The unitary semantics of a typing context Γ with respect to τ, called the *unitary semantics* of Γ and written $[\![\Gamma]\!]_\tau$, is defined as

$$[\![\Gamma]\!]_\tau \triangleq \left\{\sigma \text{ substitution} : \mathrm{dom}(\sigma) = \mathrm{dom}(\Gamma) \text{ and } \forall x \in \mathrm{dom}(\sigma), \sigma(x) \in [\![\Gamma(x)]\!]_\tau\right\}.$$

We define $\vec{t}\langle\sum_i \alpha_i \cdot v_i/x\rangle \triangleq \sum_i \alpha_i \cdot \vec{t}[v_i/x]$. The notation $\vec{t}\langle\sigma\rangle$ is also used for any substitution σ. We say that a typing judgment $\Gamma;\Delta \vdash \vec{t} : A$ is *valid* when $\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma,\Delta)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all σ and τ such that $\sigma \in [\![\Gamma,\Delta]\!]_\tau$. A typing rule is valid when the validity of its premises implies the validity of its conclusion.

Lemma 5.2. *Let $A, B \in \mathbb{T}$ and τ defined in $FV(A,B)$. Then, $[\![A[B/X]]\!]_\tau = [\![A]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}}$.*

Proof. By induction on $A \in \mathbb{T}$. The full proof is given in Appendix A.2. □

Theorem 5.3. *The typing rules in Figure 6 are valid.*

Proof. We have to check each rule with respect to the definition of typing judgement. To exemplify, we give the case of rule $(\multimap_i)$. The proof for the remaining rules is given in Appendix A.2.

Suppose that $\Gamma;\Delta,x : A \vdash \vec{t} : B$ is a valid judgment. Then since $(\mathrm{dom}(\Delta) \cup \{x\}) \subseteq FV(\vec{t})$ we have $\mathrm{dom}(\Delta) \subseteq FV(\lambda x.\vec{t})$. Similarly, since $FV(\vec{t}) \subseteq (\mathrm{dom}(\Gamma,\Delta) \cup \{x\})$ it is also true that $FV(\lambda x.\vec{t}) \subseteq \mathrm{dom}(\Gamma,\Delta)$. For any $\sigma \in [\![\Gamma,\Delta]\!]_\tau$, we have that $(\lambda x.\vec{t})\langle\sigma\rangle = \lambda x.\vec{t}\langle\sigma\rangle$, since $x \notin \mathrm{dom}(\sigma)$. For all $\vec{v} \in [\![A]\!]_\tau$, we observe that $(\vec{t}\langle\sigma\rangle)\langle\vec{v}/x\rangle = \vec{t}\langle\sigma,\{\vec{v}/x\}\rangle \Vdash [\![B]\!]_\tau$, since $\sigma,\{\vec{v}/x\} \in [\![\Gamma,x : A]\!]_\tau$. Therefore, $(\lambda x.\vec{t})\langle\sigma\rangle \Vdash [\![A \multimap B]\!]_\tau$. □

5.2 Isometries over Qubits

Definition 5.4 (Isometry and unitarity). An operator $\mathcal{F} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$ is *isometric* if it preserves the inner product, i.e., if $\langle\mathcal{F}(u)|\mathcal{F}(v)\rangle = \langle u|v\rangle$, $\forall u, v \in \overline{\mathbb{Q}}^{2^n}$. Furthermore, if $\mathcal{F} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^n}$ is isometric then $\mathcal{F}$ is *unitary*.

To simplify notation, for $i = 0,\dots,2^n-1$, let $|i\rangle \triangleq (|i_1\rangle,(|i_2\rangle,\dots(|i_{n-1}\rangle,|i_n\rangle)\dots))$, where $i_1\dots i_n$ is the binary representation of i. We show that any linear application between qubit terms must preserve orthogonality and the norm of basis states.

Lemma 5.5. *For any $\vec{t} \in$ PUNQ and any $k \geq n$ for $k,n \in \mathbb{N} - \{0\}$,*

$$\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset \iff \begin{cases}\forall i = 0,\dots,2^n-1, \exists \vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset \text{ such that } \vec{t}\,|i\rangle \rightsquigarrow^* \vec{v}_i, \text{ and}\\ \vec{v}_i \perp_{\sharp(\mathbb{B}^k)} \vec{v}_j,\ \forall i \neq j.\end{cases}$$

Proof. The proof is given in Appendix A.2. □

This property will allow us to show that closed superpositions of qubit applications behave as isometric operators. Let $\lfloor - \rfloor$ be the linear map from closed values of type $\sharp(\mathbb{B}^n)$ to $\overline{\mathbb{Q}}^{2^n}$ defined by $\lfloor\sum_{i=0}^{2^n-1}\alpha_i\cdot|i\rangle\rfloor \triangleq \begin{pmatrix}\alpha_0 & \cdots & \alpha_{2^n-1}\end{pmatrix}^T \in \overline{\mathbb{Q}}^{2^n}$. A closed superposition $\vec{t} \in \mathsf{S}_c$ of type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ *represents* an operator $\mathcal{F} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$ if, for all values $\vec{v}$ of type $\sharp(\mathbb{B}^n)$ and $\vec{w}$ of type $\sharp(\mathbb{B}^k)$, it holds that $\vec{t}\,\vec{v} \rightsquigarrow^* \vec{w}$ if and only if $\mathcal{F}(\lfloor\vec{v}\rfloor) = \lfloor\vec{w}\rfloor$.

By linearity of $\multimap$, preserving the norm of the basis states is enough to show that an abstraction represents an isometry. Moreover, when the dimensions n and k match the abstraction represents a unitary operator.

Theorem 5.6 (Soundness). *For any closed superposition* $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$ *and any* $k, n \in \mathbb{N} - \{0\}$*, the following hold:*

(1) If $k \geq n$*, then* $\lambda x.\vec{t}$ *represents an isometry in* $\overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$*,*

(2) If $k = n$*, then* $\lambda x.\vec{t}$ *represents a unitary operator in* $\overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^n}$*.*

Furthermore, if $k < n$*, then the type is uninhabited.*

Proof. Let $\|\cdot\|$ and $\cdot^\dagger$ represent the typical vector norm and the conjugate transpose, respectively.

1) Let $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$. Then, from Lemma 5.5, for $i = 0, \ldots, 2^n - 1$, there exist $\vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$ such that $\vec{t}[|i\rangle/x] \rightsquigarrow^* \vec{v}_i$ with $\langle \vec{v}_i | \vec{v}_j \rangle = 0$, for $i \neq j$. Let $\mathcal{V} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$ be the linear operator defined by $\mathcal{V}(\lfloor |i\rangle \rfloor) = \lfloor \vec{v}_i \rfloor$, for each $i = 0, \ldots, 2^n - 1$. By the linearity of the calculus we have that $\lambda x.\vec{t}$ represents the operator $\mathcal{V}$. Moreover, $\mathcal{V}$ is isometric by Definition 5.4 since $\|\lfloor \vec{v}_i \rfloor\| = 1$ and $\lfloor \vec{v}_i \rfloor^\dagger \cdot \lfloor \vec{v}_j \rfloor = 0$, for $0 \leq i \neq j \leq 2^n - 1$.

2) The case $(k = n)$ is easy to show, since from the case $(k \geq n)$ the term $\lambda x.\vec{t}$ must represent some isometric operator that in this case is also dimension-preserving, so by Definition 5.4 it must also be unitary. Let us now show that the type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ is uninhabited for $k < n$. We will consider an argument by contradiction. Let us start with a typable and closed abstraction $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$. Then, let us define the values $\vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$ satisfying $(\lambda x.\vec{t})\ |i\rangle \rightsquigarrow^* \vec{v}_i$. Since there are 2^n such $\vec{v}_i$, they must not be linearly independent, meaning that there is a choice of $\beta_i \in \overline{\mathbb{Q}}$ such that $\sum_{i=1}^{2^n} \beta_i \cdot \vec{v}_i = \vec{0}$ and not all values β_i are zero. Then, we may define the superposition $\vec{s} \triangleq \sum_i \frac{\beta_i}{\sqrt{\sum_i |\beta_i|^2}} \cdot |i\rangle \in [\![\sharp(\mathbb{B}^n)]\!]_\emptyset$ for which $(\lambda x.\vec{t})\ \vec{s} \rightsquigarrow^* \vec{0} \notin [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$. This concludes the proof. □

Notice that, since we have that $[\![\S A]\!]_\tau \triangleq [\![A]\!]_\tau$ in the unitary semantics, the result of Theorem 5.6 also applies to terms $\lambda x.\vec{t}$ obtained from duplication and therefore having a type that contains the § symbol, since $[\![\S^a \sharp(\mathbb{B}^n) \multimap \S^b \sharp(\mathbb{B}^k)]\!]_\emptyset = [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$.

Theorem 5.7 (Completeness). *For any isometry* $\mathcal{I} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$*, with* $k \geq n \geq 1$*, there exists a closed superposition* $\lambda x.\vec{t} \in \mathsf{S}_c$ *of type* $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ *that represents* $\mathcal{I}$*.*

Proof. Constructive proof given in Appendix A.2. □

On top of satisfying unitarity, PUNQ also satisfies another natural property of quantum computation, which is that qubits may occupy entangled states and that there is no operation that disentangles any general state.

Theorem 5.8 (Non-separability). *For any* $n, k \in \mathbb{N} - \{0\}$*, the type* $\sharp(\mathbb{B}^{n+k}) \multimap (\sharp(\mathbb{B}^n) \times \sharp(\mathbb{B}^k))$ *is uninhabited.*

Proof. Proof in Appendix A.2. □

6 POLYTIME NORMALIZATION

In this section, we show that PUNQ preserves the polytime strong normalization properties of DLAL [Baillot and Terui 2009], when adapted to the context of quantum computation. For that purpose, we define an encoding from superpositions in PUNQ to sets of DLAL terms and show that the polynomial time strong normalization of each of the terms in this set implies the polytime normalization of the initial PUNQ program. All the proofs omitted in this section are fully developed in Appendix A.3.

$$\frac{\Gamma;\Delta \vdash t : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t : A}\ (\text{Weak}) \qquad \frac{\Gamma, x : B, y : B;\Delta \vdash t : A}{\Gamma, x : B;\Delta \vdash t[x/y] : A}\ (\text{Cntr}) \qquad \frac{}{;x : A \vdash x : A}\ (\text{Id})$$

$$\frac{\Gamma;\Delta, x : A \vdash t : B}{\Gamma;\Delta \vdash \lambda x.t : A \multimap B}\ (\multimap \text{i}) \qquad \frac{\Gamma;\Delta \vdash t : A \multimap B \quad \Gamma';\Delta' \vdash s : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t\ s : B}\ (\multimap \text{e})$$

$$\frac{\Gamma, x : A;\Delta \vdash \vec{t} : B}{\Gamma;\Delta \vdash \lambda x.\vec{t} : A \Rightarrow B}\ (\Rightarrow \text{i}) \qquad \frac{\Gamma;\Delta \vdash t : A \Rightarrow B \quad ;[z : C] \vdash s : A}{\Gamma, [z : C] ;\Delta \vdash t\ s : B}\ (\Rightarrow \text{e})$$

$$\frac{;\Gamma,\Delta \vdash t : A}{\Gamma;\S\Delta \vdash t : \S A}\ (\S \text{ i}) \qquad \frac{\Gamma;\Delta \vdash s : \S B \quad \Gamma';\Delta', x : \S B \vdash t : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t[s/x] : A}\ (\S \text{ e})$$

$$\frac{\Gamma;\Delta \vdash t : A \quad X \notin FV(\Gamma,\Delta)}{\Gamma;\Delta \vdash t : \forall X.A}\ (\forall \text{ i}) \qquad \frac{\Gamma;\Delta \vdash t : \forall X.A}{\Gamma;\Delta \vdash t : A[B/X]}\ (\forall \text{ e})$$

Fig. 10. Dual Light Affine Logic (DLAL).

6.1 Dual Light Affine Logic

DLAL terms [Baillot and Terui 2009] are a strict subset of System F [Girard 1972; Reynolds 1974] that can be typed with respect to the rules of Figure 10. The set $\mathbb{S}$ of types in DLAL is the set produced by the following grammar:

$$A := X \mid A \multimap A \mid A \Rightarrow A \mid \S A \mid \forall X.A,$$

and can be viewed as a subset of $\mathbb{T}$. The notion of typing environments Γ, Δ are defined in a standard way. For a given set of DLAL terms $\mathcal{S}$, we write $\Gamma;\Delta \vdash_{\text{DLAL}} \mathcal{S} : A$ if $\forall t \in \mathcal{S}$, the derivation of $\Gamma;\Delta \vdash t : A$ can be obtained in DLAL.

The size of a DLAL term t, denoted $|t|$, is defined as the number of its symbols, and the size of a set of terms $\mathcal{S}$, noted $|\mathcal{S}|$, is defined as the biggest size of a term $t \in \mathcal{S}$, i.e., $|\mathcal{S}| \triangleq \max\{|t| \mid t \in \mathcal{S}\}$. The reason for this choice is that superpositions in PUNQ will be encoded as sets of terms in DLAL, which in a sense will encode the set of base terms present in the superposition. Since in PUNQ the size of a term is the maximum size appearing in a superposition, this choice ensures that we capture the correct notion of polynomial size of the final term.

The following theorem implies both a polynomial bound on the reduction length and a polynomial bound on the size of each term obtained during the reduction in the DLAL system.

Theorem 6.1 (Polynomial time strong normalization [Baillot and Terui 2009, Thm. 8]). *If $\Gamma;\Delta \vdash_{\text{DLAL}} t : A$, then there exists a constant d, referred to as the* depth of t*, such that t reduces to its normal form in at most $O(|t|^{2^d})$ reduction steps and within a time complexity of $O(|t|^{2^{d+2}})$ on a deterministic Turing machine. This result holds independently of the chosen reduction strategy.*

Given two sets of DLAL terms $\mathcal{S}$ and $\mathcal{T}$, we write $\mathcal{S} \to \mathcal{T}$ if $\mathcal{T} = \{t \mid s \in \mathcal{S} \wedge (s \to t \vee s \nrightarrow)\}$, meaning that the set $\mathcal{T}$ is obtained from $\mathcal{S}$ by attempting to apply a single reduction step. Therefore, if the term is already normalized it remains in the set. Let $\to^n$ be the n-fold transitive closure of $\to$, i.e., if $\mathcal{S}_0 \to \ldots \to \mathcal{S}_n$ then $\mathcal{S}_0 \to^n \mathcal{S}_n$. We also write $\mathcal{S} \to^{\leq n} \mathcal{T}$ if there exists $k \leq n$ such that $\mathcal{S} \to^k \mathcal{T}$. Termination is defined as reaching a point where all terms in the set are in normal form, i.e., $\mathcal{S} \downarrow^n$ if $\mathcal{S} \to^n \mathcal{T}$ and $\mathcal{T} \to \mathcal{T}$. Similarly, we write $\mathcal{S} \downarrow^{\leq n}$ for termination in at most n steps.

Corollary 6.2. *For any type $A \in \mathbb{T}$, there is a polynomial $P_A \in \mathbb{N}[X]$ such that, if $\Gamma; \Delta \vdash_{\mathrm{DLAL}} \mathcal{S} : A$, then $\mathcal{S} \downarrow^{\leq P_A(|\mathcal{S}|)}$ and, for each $\mathcal{T}$ satisfying $\mathcal{S} \to^n \mathcal{T}$ for some $n \in \mathbb{N}$, $|\mathcal{T}| \leq P_A(|\mathcal{S}|)$.*

Notice that the polynomial P_A can be explicitly computed and only depends on A as the depth of a term is a function of its type (see [Baillot and Terui 2009]). Furthermore, Corollary 6.2 applies for any reduction strategy, so we may fix the strategy to be call-by-value for our purposes.

6.2 Encoding of PUNQ in DLAL

In order to prove PUNQ's polynomial time normalization, we will give a translation from PUNQ terms into sets of DLAL terms. We will then show that, for the PUNQ terms of interest (i.e. those that are typable), their reduction length in PUNQ is upperbounded by the maximum reduction length in its translation. Given that this encoding will only increase polynomially the size of the term, this will provide a polynomial upper bound in the reduction of PUNQ terms.

We extend the DLAL syntax with the term $*$, which will represent the encoding of $\vec{0}$, and the following typing rule ensuring that $*$ has any type:

$$\frac{}{;\vdash * : \forall X.X}\ (*)$$

We will call this extension of the DLAL syntax and typing system DLAL_*. Since $*$ does not reduce in any way, the polytime termination result in Theorem 6.1 and its corollary also apply to DLAL_*.

Given two sets of terms $\mathcal{S}, \mathcal{T}$, let $\mathcal{S}\ \mathcal{T}$ be syntactic sugar for the set $\{s\ t \mid s \in \mathcal{S} \wedge t \in \mathcal{T}\}$. We now define formally the map $(\!|\cdot|\!) : \mathrm{PUNQ} \to \mathcal{P}(\mathrm{DLAL}_*)$ in Figure 11 and the type encoding $(\cdot)^\star : \mathbb{T} \to \mathbb{S}$ in Figure 12. This encoding is extended to contexts as follows: $\Gamma^\star \triangleq \{x_i : (A_i)^\star \mid x_i : A_i \in \Gamma\}$.

Such an encoding does not require that the set of DLAL terms perfectly simulate the corresponding PUNQ term, but it must contain enough of the structure of the original term to provide a valid upper bound in its number of reduction steps. As such, the type encoding $(\cdot)^\star$ in Figure 12 can be seen as a "dequantization" of the type (i.e. removing quantum superposition) combined with translation into DLAL types that correspond to the basic PUNQ types, such as the encoding of pairs and booleans.

The term translation $(\!|\cdot|\!)$ from PUNQ terms to sets of DLAL_* terms is standard, with a few exceptions. In DLAL, a term encoding the conditional "if t then t_1 else t_2" would be typed multiplicatively, requiring disjoint contexts for t_1 and t_2. However, for the purpose of guaranteeing unitarity, the conditional is typed additively in PUNQ (i.e., the two contexts are the same). To handle this difference, we rewrite the conditional such that the branches have no open linear variables, which allows for a correct typing derivation in DLAL_*. This transformation in general requires that we use information on the derivation of the term, but this is handled implicitly in the translation. For some natural number k, we define the notation $\lambda x_{[1..k]}.\vec{t} \triangleq \lambda x_1.\lambda x_2.\ldots.\lambda x_k.\vec{t}$. Similarly, when $\vec{t}$ contains k instances of a free variable x, we denote by $\vec{t}_{[x:1..k]}$ the term $\vec{t}$ where each occurrence of x is replaced by x_i, for $i = 1, \ldots, k$. For instance, for $\vec{t} = \lambda y.x(x\ y)$, we have $\vec{t}_{[x:1..k]} = \lambda y.x_1(x_2\ y)$.

Also of note is the translation of the abstraction "$\lambda x.\vec{t}$", where we perform a *linearization* of the lambda abstraction, separating all the repeated instances of x with separate variables, labeled x_i. Likewise, in the application case "$t_1\ t_2$", we repeat the encoding of t_2 the appropriate number of times. This is encoded in a function $\eta : \mathrm{PUNQ} \to \{0, 1, \ldots\}$, encoding the number of times the first input is repeated. This ensures that generated set of DLAL terms will account for all possible branches contained in the original PUNQ term (see Example 6.11). For instance, $\eta(\lambda x.x) = 1$ since x is not duplicated, and $\eta((\lambda x.x)(\lambda f.\lambda y.f\ f\ y)) = 2$ since once the term reduces to the value $\lambda f.\lambda y.f\ f\ y$, the input f appears twice in the term.

$$(\!|\vec{0}|\!) \triangleq \{*\} \qquad (\!|x|\!) \triangleq \{x\} \qquad (\!|\,|0\rangle\,|\!) \triangleq \{\lambda x.\lambda y.x\} \qquad (\!|\,|1\rangle\,|\!) \triangleq \{\lambda x.\lambda y.y\}$$

$$(\!|\,\text{if } t \text{ then } t_1 \text{ else } t_2\,|\!) \triangleq \Big((\!|t|\!)\ \lambda x_{[1..k]}.(\!|t_1|\!)\ \lambda x_{[1..k]}.(\!|t_2|\!)\Big)\ x_1 \dots x_k, \quad \text{with } \Gamma; x_1 : A_1, \dots, x_k : A_k \vdash t_1, t_2 : B$$

$$(\!|\,\text{let } (x,y) = t_1 \text{ in } t_2\,|\!) \triangleq (\!|t_1|\!)\ (\lambda x.\lambda y.(\!|t_2|\!)) \qquad (\!|\alpha \cdot \vec{t}|\!) \triangleq (\!|\vec{t}|\!), \text{ if } \alpha \neq 0, \{*\} \text{ otherwise}$$

$$(\!|\vec{t_1} + \vec{t_2}|\!) \triangleq (\!|\vec{t_1}|\!) \cup (\!|\vec{t_2}|\!) \qquad (\!|t_1\ t_2|\!) \triangleq \big(\dots\big((\!|t_1|\!)\ (\!|t_2|\!)\big)\ (\!|t_2|\!)\ \big)\dots(\!|t_2|\!)\big) \quad (\eta(t_1) \text{ times})$$

$$(\!|\,(t_1,t_2)\,|\!) \triangleq \lambda x.(x\ (\!|t_1|\!)\ (\!|t_2|\!)) \qquad (\!|\lambda x.\vec{t}|\!) \triangleq \lambda x_{[1..k]}.(\!|\vec{t}_{[x:1..k]}|\!), \quad k = \eta(\lambda x.\vec{t})$$

Fig. 11. $(\!|\cdot|\!) : \text{PUNQ} \to \mathcal{P}(\text{DLAL}_*)$.

$$(X)^\star \triangleq X \qquad (\mathbb{B})^\star \triangleq \forall X.(X \multimap X \multimap X)$$
$$(A_1 \multimap A_2)^\star \triangleq (A_1)^\star \multimap (A_2)^\star \qquad (A_1 \Rightarrow A_2)^\star \triangleq (A_1)^\star \Rightarrow (A_2)^\star$$
$$(A_1 \times A_2)^\star \triangleq \forall X.(((A_1)^\star \multimap (A_2)^\star \multimap X) \multimap X) \qquad (\sharp Q)^\star \triangleq (Q)^\star$$
$$(\S A)^\star \triangleq \S(A)^\star \qquad (\forall X.A)^\star \triangleq \forall X.(A)^\star$$

Fig. 12. $(\cdot)^\star : \mathbb{T} \to \mathbb{S}$.

Let us now look at a few properties of the type encoding $(\cdot)^\star$. A first property we may notice is that the type encoding does not change if we first transform a type into its clonable version, i.e. when we apply the ! function.

Lemma 6.3. *For any $A \in \mathbb{T}$, we have that $(!(A))^\star = A^\star$.*

Proof. By induction on the structure of A. □

Similarly, a useful fact is that the encoding is preserved by PUNQ subtyping.

Lemma 6.4. *For any $A, B \in \mathbb{T}$, if $A \leq B$ then $A^\star = B^\star$.*

Proof. By inspection of the rules in Figure 5. □

Regarding the term translation $(\!|\cdot|\!)$, we can check that it is preserved by the equivalence relation $\equiv$ in PUNQ, modulo the presence of the term $*$.

Definition 6.5. Let $\mathcal{S}, \mathcal{T}$ be two sets. Then $\mathcal{S} \sqsubseteq \mathcal{T}$ if $\mathcal{S} \subseteq \mathcal{T} \cup \{*\}$. We also define $\mathcal{T} \simeq \mathcal{S}$ if $\mathcal{T} \sqsubseteq \mathcal{S}$ and $\mathcal{S} \sqsubseteq \mathcal{T}$.

Lemma 6.6. *If $\vec{t} \equiv \vec{r}$, then $(\!|\vec{t}|\!) \simeq (\!|\vec{r}|\!)$.*

Proof. By inspection of the equivalence relation. The full proof is given in Appendix A.3. □

We now turn to the normalization behavior of each term in the set corresponding to a given translation. First, we may notice that every term in a translation has the same type, which corresponds to the type given by the $(\cdot)^\star$. Notice also that, though not explicitly stated, the notion of depth of a typing derivation [Baillot and Terui 2009] is asymptotically preserved by the translation.

Lemma 6.7. $\Gamma; \Delta \vdash \vec{t} : A$ *implies* $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!) : A^\star$.

Proof. By induction on the derivation of $\Gamma; \Delta \vdash \vec{t} : A$. The full proof is given in Appendix A.3. □

Another important detail is that the encoding increase the size of the original PUNQ term only polynomially and is therefore an appropriate way to connect the complexity of the two systems.

Lemma 6.8. *For any $A \in \mathbb{T}$, there exists a polynomial P_A such that, for any $\vec{t}$ where $; \vdash \vec{t} : A$, the following holds $|(\!|\vec{t}|\!)| = O(P_A(|\vec{t}|))$.*

Proof. By induction on the structure of $\vec{t}$. The full proof is given in Appendix A.3. □

To demonstrate how this translation guarantees polynomial-time termination in PUNQ, we consider the relation between the reduction of a PUNQ term $\vec{t}$ and the reduction of its translation $(\!|\vec{t}|\!)$. Given our choice of translation, a term in PUNQ will correspond to a set in DLAL containing not only all basis states present in any superposition, but also all possible branchings in the program.

We now show that a reduction in PUNQ corresponds to at least one reduction over its encoding in DLAL. Notice first that, once we reach a value in PUNQ, its DLAL translation will not reduce.

Lemma 6.9. *For any $\vec{v} \in \mathsf{V}_c$, we have that $(\!|\vec{v}|\!)$ does not reduce.*

Proof. By induction on the structure of $\vec{v}$. The full proof is given in Appendix A.3. □

Now we consider the central lemma for the proof of Theorem 6.12.

Lemma 6.10. *For any $\vec{t} \in$ PUNQ, $\vec{t} \rightsquigarrow \vec{r}$ implies $(\!|\vec{r}|\!) \sqsubseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|\vec{t}|\!) \to^n \mathcal{S}_n\}$.*

Proof. By induction on relation $\rightsquigarrow$. The full proof is given in Appendix A.3. □

Example 6.11 (Two Hadamards). Let $\vec{t} = (\underline{2}\ \mathsf{H})\ |0\rangle$, having PUNQ type $\S\sharp\mathbb{B}$, correspond to applying twice the Hadamard gate (Example 4.2) to state $|0\rangle$. We denote by $\mathcal{S}_0$ its translation into a set of DLAL terms, and, for all $n > 0$, let $\mathcal{S}_n$ be the set such that $\mathcal{S}_0 \to^n \mathcal{S}_n$. Let $\mathtt{tt} \triangleq \lambda x.\lambda y.x$ and $\mathtt{ff} \triangleq \lambda x.\lambda y.y$ be a shorthand notation for the boolean encoding in DLAL. First, we translate the Hadamard term. Notice that, since $|+\rangle$ and $|-\rangle$ are closed terms, we do not need to add any new variables in the translation of the if $(\cdot)$ then $(\cdot)$ else $(\cdot)$. The translation is then:

$$(\!|\,\mathsf{H}\,|\!) = \lambda x.(\!|\,\mathsf{if}\ x\ \mathsf{then}\ |+\rangle\,\mathsf{else}\ |-\rangle\,|\!) = \lambda x.x\ (\!|\,|+\rangle\,|\!)\ (\!|\,|-\rangle\,|\!) = \lambda x.x\ \{\mathtt{ff},\mathtt{tt}\}\ \{\mathtt{ff},\mathtt{tt}\},$$

which corresponds to the set of 4 terms $\{\lambda x.x\ b_1\ b_2 \mid b_1, b_2 \in \{\mathtt{ff},\mathtt{tt}\}\}$. The translation of $\vec{t}$ corresponds to 16 DLAL terms, all of which have type $(\S\sharp\mathbb{B})^\star = \S(\forall X.X \multimap X \multimap X)$ in DLAL$_*$:

$$(\!|\vec{t}|\!) = \mathcal{S}_0 = \left\{(\lambda f_1.\lambda f_2.\lambda q.f_1\,(f_2\ q)\ (\lambda x.x\ b_1\ b_2)\ (\lambda x.x\ b_3\ b_4))\ \mathtt{tt} \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}.$$

This set reduces in 7 steps, according to the following progression:

$$\begin{aligned}
\mathcal{S}_1 &= \left\{(\lambda f_2.\lambda q.(\lambda x.x\ b_1\ b_2)(f_2\ q)\ (\lambda x.x\ b_3\ b_4))\ \mathtt{tt} \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_2 &= \left\{(\lambda q.(\lambda x.x\ b_1\ b_2)\ (\lambda x.x\ b_3\ b_4)\ q)\ \mathtt{tt} \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_3 &= \left\{(\lambda x.x\ b_1\ b_2)\ (\lambda x.x\ b_3\ b_4)\ \mathtt{tt} \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_4 &= \left\{(\lambda x.x\ b_1\ b_2)\ (\mathtt{tt}\ b_3\ b_4) \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_5 &= \left\{(\lambda x.x\ b_1\ b_2)\ b_3 \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_6 &= \left\{b_3\ b_1\ b_2 \mid b_i \in \{\mathtt{ff},\mathtt{tt}\}\right\}\\
\mathcal{S}_7 &= \{\mathtt{ff},\mathtt{tt}\}
\end{aligned}$$

In PUNQ, $\vec{t}$ reduces to a value in 6 steps. Defining $\vec{r}_i$ as $\vec{r}_0 \triangleq \vec{t}$ and $\vec{r}_i \rightsquigarrow \vec{r}_{i+1}$, we have

$$\begin{array}{ll}
\vec{r}_1 = (\lambda x.\mathsf{H}\ \mathsf{H}\ x)\ |0\rangle & (\!|\vec{r}_1|\!) = \mathcal{S}_2 \\
\vec{r}_2 = \mathsf{H}\ \mathsf{H}\ |0\rangle & (\!|\vec{r}_2|\!) = \mathcal{S}_3 \\
\vec{r}_3 = \mathsf{H}\ (\mathsf{if}\ |0\rangle\,\mathsf{then}\ |+\rangle\,\mathsf{else}\ |-\rangle) & (\!|\vec{r}_3|\!) = \mathcal{S}_4 \\
\vec{r}_4 = {}^1\!/\!_{\sqrt{2}} \cdot \mathsf{H}\ |0\rangle + {}^1\!/\!_{\sqrt{2}} \cdot \mathsf{H}\ |1\rangle & (\!|\vec{r}_4|\!) = \mathcal{S}_5 \\
\vec{r}_5 = {}^1\!/\!_{\sqrt{2}} \cdot \mathsf{if}\ |0\rangle\,\mathsf{then}\ |+\rangle\,\mathsf{else}\ |-\rangle + {}^1\!/\!_{\sqrt{2}} \cdot \mathsf{if}\ |1\rangle\,\mathsf{then}\ |+\rangle\,\mathsf{else}\ |-\rangle & (\!|\vec{r}_5|\!) = \mathcal{S}_6 \\
\vec{r}_6 = |0\rangle & (\!|\vec{r}_6|\!) = \{\mathtt{tt}\} \subseteq \mathcal{S}_7
\end{array}$$

Notice that the PUNQ term was not perfectly simulated since $\mathtt{ff}$ also appears in $\mathcal{S}_7$. However, in every step of the reduction $\vec{r}_0 \rightsquigarrow^* \vec{r}_6$, we find the translation of $\vec{r}_i$ in a further reduction of $\mathcal{S}_0$, which suffices to bound the number of steps until achieving normalization.

The previous results allow us to conclude that, like DLAL, PUNQ ensures polytime normalization.

Theorem 6.12 (Polynomial time normalization). *For any $A \in \mathbb{T}_c$, there is a polynomial $P_A \in \mathbb{N}[X]$ such that, for any* PUNQ *superposition $\vec{t}$ with type A, $\vec{t}$ reduces to a value $\vec{v}$ in at most $O(P_A(|\vec{t}|)$ reduction steps and $|\vec{v}| = O(P_A(|\vec{t}|))$.*

Proof. Let $\vec{t} \in$ PUNQ such that $\Gamma; \Delta \vdash \vec{t} : A$. Then, by Lemma 6.7, $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}_*} (\!|\vec{t}|\!) : A^\star$ holds and each term $t_i \in (\!|\vec{t}|\!)$ has size at most $O(|\vec{t}|)$. By Corollary 6.2, there is a polynomial $P_{A^\star} \in \mathbb{N}[X]$ such that every DLAL_* term $t_i \in (\!|\vec{t}|\!)$ can be reduced in $N \triangleq O(P_{A^\star}(|t_i|))$ steps. Since the size $|t_i|$ of DLAL_* terms is bounded by $P_A(|\vec{t}|)$, from Lemma 6.8, we have $N = O(P_{A^\star}(P_A(|\vec{t}|)))$. Therefore all the terms in $(\!|\vec{t}|\!)$ reduce in polynomial time on the size of $\vec{t}$. As shown in Lemma 6.10, each reduction step in PUNQ corresponds to at least one reduction step applied to all the (non-normalized) terms in its translation into DLAL_*, we conclude that $\vec{t}$ must reduce to normal form in at most N steps. □

7 TYPE CHECKING

In this section we discuss type checking in PUNQ, which corresponds to the following problem.

Definition 7.1 (Type checking). We define the problem of type checking in PUNQ, denoted CHK, as follows. Given $\vec{t} \in \mathsf{S}_c$ and $A \in \mathbb{T}_c$, $\mathsf{CHK}(\vec{t}, A)$ is true if and only if $; \vdash \vec{t} : A$ can be derived.

Preliminaries. Let PTIME be the set of decision problems solvable by a deterministic Turing machine running in polynomial time and, given an oracle C, let us denote by PTIME^C the set of decision problems that can be determined by a polytime deterministic oracle Turing machine with oracle C, which can be easily extended to the case where C is a complexity class. Finally, the class Π_1^0 of the arithmetical hierarchy is defined as $\Pi_1^0 \triangleq \left\{\psi \mid \exists \phi \in \mathsf{REC}, \forall \overline{x}.(\psi(\overline{x}) \Leftrightarrow \forall \overline{y}.\phi(\overline{x}, \overline{y}))\right\}$, where $\overline{x}, \overline{y}$ are variables with values belonging to enumerable sets, ψ and ϕ are formulas in the language of first-order arithmetic, and REC is the class of recursive sets (decidable problems) [Odifreddi 1992].

Orthogonality criteria. Outside of checking orthogonality in rules ($\mathrm{if}_\sharp$) and ($\sharp_i$), PUNQ type checking is no more complicated than that of DLAL, which is decidable in polynomial time [Baillot and Terui 2009]. This is because the complexity added by the Lambda-$\mathcal{S}_1$ typing discipline (modulo orthogonality checking) corresponds to that of a simply typed lambda calculus, where type checking can be done efficiently [Mairson 2004]. Therefore, if one has access to an oracle for checking orthogonality between terms, then type checking in PUNQ can be done efficiently.

Definition 7.2. Let φ be an *oracle for orthogonality* between PUNQ terms, where, for all $\vec{t}, \vec{s} \in \mathsf{S}$, and contexts Γ, Δ, we have $\varphi(\vec{t}, \vec{s}, \Gamma, \Delta) \triangleq 0 \iff \vec{t} \perp^{\Gamma;\Delta} \vec{s}$.

Lemma 7.3. $\mathsf{CHK} \in \mathsf{PTIME}^{\varphi}$.

We will address the complexity of type checking in three fragments of the PUNQ language, defined by their orthogonality criteria. The first two fragments only require access to the syntax of the term, whereas the last one takes in as input a "candidate type" used to bound the time needed to check orthogonality. Notice that, since we only check orthogonality in cases where we also check the type – rules ($\mathrm{if}_\sharp$) and ($\sharp_i$), this does not lead to a loss of generality.

Definition 7.4. We define the following three criteria for orthogonality over superpositions.

- $\mathsf{ORTHO}_\emptyset : \mathsf{S} \times \mathsf{S} \to \{0, 1\}$ defined as

$$\mathsf{ORTHO}_\emptyset(\vec{t}, \vec{s}) \triangleq \begin{cases} 0, & \text{if } \vec{t} \perp \vec{s}, \\ 1, & \text{otherwise.} \end{cases}$$

- $\mathsf{ORTHO}_\times : \mathsf{S} \times \mathsf{S} \to \{0, 1\}$ defined as

$$\mathsf{ORTHO}_\times(\vec{t}, \vec{s}) \triangleq \begin{cases} \mathsf{ORTHO}_\times(t_1, s_1) \cdot \mathsf{ORTHO}_\times(t_2, s_2), & \text{if } \vec{t} = (t_1, t_2) \text{ and } \vec{s} = (s_1, s_2), \\ \mathsf{ORTHO}_\emptyset(\vec{t}, \vec{s}), & \text{otherwise.} \end{cases}$$

- $\mathsf{ORTHO}_{\mathsf{untyped}} : \mathsf{S} \times \mathsf{S} \times \mathbb{T} \to \{0, 1\}$ defined as follows. Let $FV(\vec{t}) \cup FV(\vec{s}) = x_1, \ldots, x_n$.

$$\mathsf{ORTHO}_{\mathsf{untyped}}(\vec{t}, \vec{s}, Q) \triangleq 1 - \prod\nolimits_{\vec{v}_1, \ldots \vec{v}_n \in \mathsf{V}_c} \mathsf{PERP}\Big(\vec{t}\langle \vec{v}_1/x_1, \ldots, \vec{v}_n/x_n\rangle,\ \vec{s}\langle \vec{v}_1/x_1, \ldots, \vec{v}_n/x_n\rangle, Q\Big),$$

with $\mathsf{PERP} : \mathsf{S}_c \times \mathsf{S}_c \times \mathbb{T} \to \{0, 1\}$ defined as

$$\mathsf{PERP}(\vec{t}_1, \vec{t}_2, Q) \triangleq 1 \iff \vec{t}_1, \vec{t}_2 \rightsquigarrow^* \vec{w}_1, \vec{w}_2 \in [\![Q]\!]_\emptyset \text{ and } \mathsf{ORTHO}_\emptyset(\vec{w}_1, \vec{w}_2) = 0.$$

For $\mathcal{X} \in \{\emptyset, \times, \mathsf{untyped}\}$, we denote by $\mathsf{CHK}_\mathcal{X}$ set of terms that can be typed in PUNQ using $\mathsf{ORTHO}_\mathcal{X}$ as definition of orthogonality.

In $\mathsf{ORTHO}_\emptyset$, orthogonality is defined only over closed terms. Notice, for instance, that the Hadamard and Z gates (Examples 4.2 and 4.3) only employ this orthogonality, and therefore $\mathsf{H}, \mathsf{Z} \in \mathsf{CHK}_\emptyset$. In $\mathsf{ORTHO}_\times$, we only detect orthogonality via closed terms in a pair structure, for instance by checking that $(|0\rangle, x)$ is orthogonal to $(|1\rangle, y)$ over any possible variable substitutions on x and y. This is used in the case of the *CNOT* gate (Example 4.4) and the circuit for teleportation which employs it (Example 4.5), and so $\mathsf{CNOT}, \mathsf{telep} \in \mathsf{CHK}_\times$. In $\mathsf{ORTHO}_{\mathsf{untyped}}$, we check over all possible value substitutions – even those where the value used does not have the correct type – check if the term reduces into the correct type and, if so, test its orthogonality.

Lemma 7.5. $\mathsf{CHK}_\emptyset \subseteq \mathsf{CHK}_\times \subseteq \mathsf{CHK}_{\mathsf{untyped}} \subseteq \mathsf{CHK}$.

$\mathsf{ORTHO}_\emptyset$ and $\mathsf{ORTHO}_\times$ can be verified efficiently, and therefore ensure that type checking can be done in polytime. However, we conjecture that general orthogonality is undecidable.

Lemma 7.6. $\mathsf{ORTHO}_\emptyset, \mathsf{ORTHO}_\times \in \mathsf{PTIME}$, *and* $\mathsf{ORTHO}_{\mathsf{untyped}} \in \Pi^0_1$.

Proof. Proof given in Appendix A.4. □

The proof of Lemma 7.6 rests also on the choice of $\overline{\mathbb{Q}}$ for the amplitudes in PUNQ, where comparisons can be done efficiently [Halava et al. 2005, Proposition 2.2]. Furthermore, since type checking is done classically, we must bound the complexity of reducing terms to check their orthogonality in the classical Turing machine model (see Appendix A.4).

Theorem 7.7. $\mathsf{CHK}_\emptyset, \mathsf{CHK}_\times \in \mathsf{PTIME}$ *and* $\mathsf{CHK}_{\mathsf{untyped}} \in \mathsf{PTIME}^{\Pi^0_1}$.

Proof. From Lemmas 7.3 and 7.6. □

8 CONCLUSIONS AND FUTURE WORK

We have introduced PUNQ, a new quantum programming language with quantum control capabilities. PUNQ stands out as the first feasible language with quantum control. Supporting higher-order programs, the language boasts a robust type system ensuring both unitarity and polynomial-time normalization. To conclude, we claim PUNQ is a pioneering language, seamlessly combining theoretical depth with practical applicability in the realm of quantum computing. Plenty of interesting questions remain open:

- *Extending quantum control.* So far, in PUNQ, we have limited superpositions to ground types, i.e., non functional types such as qubits. Extending superpositions to functional types, e.g., $\sharp(A \multimap B)$ or $\sharp(A \Rightarrow B)$, would allow for natural characterizations of experiments in indefinite causal order such as the quantum switch [Procopio et al. 2015; Rubino et al. 2017]. This question is also interesting for polymorphism, since the lists of type A in DLAL, $\forall X.(A \multimap X \multimap X) \Rightarrow \S(X \multimap X)$, cannot be superposed in PUNQ.
- *Orthogonality criteria.* Extending the tractable criteria for orthogonality given in Section 7 to allow for more expressivity in the oracle-free case.
- *Circuit compilation.* Limiting quantum programs to polynomial time in a quantum Turing machine corresponds to a limitation to uniform quantum circuits [Yao 1993]. However, direct compilation strategies that preserve polynomial time are not trivial [Hainry et al. 2023]. In this vein, it would be interesting to show a compilation strategy that would assign a polysize circuit representation of any PUNQ term that corresponds to a unitary transformation.

REFERENCES

Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. 1997. Quantum Computability. *SIAM J. Comput.* 26, 5 (1997), 1524–1540. https://doi.org/10.1137/S0097539795293639

Thorsten Altenkirch and Jonathan Grattage. 2005. A Functional Quantum Programming Language. In *LICS 2005*. IEEE Computer Society, 249–258. https://doi.org/10.1109/LICS.2005.1

Andris Ambainis. 2003. Quantum walks and their algorithmic applications. *International Journal of Quantum Information* 01, 04 (2003), 507–518. https://doi.org/10.1142/S0219749903000383

Pablo Arrighi and Gilles Dowek. 2017. Lineal: A linear-algebraic Lambda-calculus. *Logical Methods in Computer Science* 13, 1 (2017), 1–33. https://doi.org/10.23638/LMCS-13(1:8)2017

Costin Badescu and Prakash Panangaden. 2015. Quantum Alternation: Prospects and Problems. In *QPL 2015 (EPTCS, Vol. 195)*, Chris Heunen, Peter Selinger, and Jamie Vicary (Eds.). Open Publishing Association, 33–42. https://doi.org/10.4204/EPTCS.195.3

Patrick Baillot and Kazushige Terui. 2009. Light types for polynomial time computation in lambda calculus. *Information and Computation* 207, 1 (2009), 41–62. https://doi.org/10.1016/j.ic.2008.08.005

Andrew Barber and Gordon Plotkin. 1996. *Dual intuitionistic linear logic.* University of Edinburgh, Department of Computer Science, Edinburgh, UK.

Stephen J. Bellantoni and Stephen A. Cook. 1992. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity* 2 (1992), 97–110. https://doi.org/10.1007/BF01201998

Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. 1993. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters* 70, 13 (1993), 1895–1899. https://doi.org/10.1103/PhysRevLett.70.1895

Ethan Bernstein and Umesh Vazirani. 1997. Quantum Complexity Theory. *SIAM J. Comput.* 26, 5 (1997), 1411–1473. https://doi.org/10.1137/S0097539796300921

Kostia Chardonnet. 2023. *Towards a Curry-Howard Correspondence for Quantum Computation.* Ph. D. Dissertation. Université Paris-Saclay. https://theses.hal.science/tel-03959403

Kostia Chardonnet, Marc de Visme, Benoît Valiron, and Renaud Vilmart. 2022. The Many-Worlds Calculus. Preprint at arXiv. http://arxiv.org/abs/2206.10234

Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. 2003. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. ACM. https://doi.org/10.1145/780542.780552

Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. 2010. Quantum implicit computational complexity. *Theoretical Computer Science* 411, 2 (2010), 377–409. https://doi.org/10.1016/j.tcs.2009.07.045

Alejandro Díaz-Caro. 2021. A quick overview on the quantum control approach to the lambda calculus. In *Proceedings of the 16th Workshop on Logical and Semantic Frameworks with Applications (LSFA'21) (Electronic Proceedings in Theoretical Computer Science, Vol. 357)*, Mauricio Ayala-Rincón and Eduardo Bonelli (Eds.). Open Publishing Association, 1–17. https://doi.org/10.4204/EPTCS.357.1

Alejandro Díaz-Caro, Gilles Dowek, and Juan Pablo Rinaldi. 2019a. Two linearities for quantum computing in the lambda calculus. *BioSystems* 186 (2019), 104012. https://doi.org/10.1016/j.biosystems.2019.104012 Postproceedings of TPNC 2017.

Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. 2019b. Realizability in the Unitary Sphere. In *LICS 2019*. IEEE, 1–13. https://doi.org/10.1109/LICS.2019.8785834

Alejandro Díaz-Caro and Octavio Malherbe. 2022. Quantum Control in the Unitary Sphere: Lambda-$\mathcal{S}_1$ and its Categorical Model. *Logical Methods in Computer Science* 18, 3 (2022), 1–32. https://doi.org/10.46298/lmcs-18(3:32)2022

Alejandro Díaz-Caro and Octavio Malherbe. 2023. A concrete model for a typed linear algebraic lambda calculus. To appear in Mathematical Structures in Computer Science. https://arxiv.org/abs/1806.09236

Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. 2008. A logical account of PSPACE. In *POPL 2008*. ACM, 121–131. https://doi.org/10.1145/1328438.1328456

Jean-Yves Girard. 1972. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. https://api.semanticscholar.org/CorpusID:117631778

Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

Jean-Yves Girard. 1998. Light Linear Logic. *Information and Computation* 143, 2 (1998), 175–204. https://doi.org/10.1006/inco.1998.2700

Jean-Yves Girard and Yves Lafont. 1987. Linear Logic and Lazy Computation. In *TAPSOFT'87 (Lecture Notes in Computer Science, Vol. 250)*. Springer, 52–66. https://doi.org/10.1007/BFb0014972

Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *PLDI 2013*. ACM, 333–342. https://doi.org/10.1145/2491956.2462177

Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

Yuri Gurevich. 1983. Algebras of Feasible Functions. In *FOCS 1983*. IEEE Computer Society, 210–214. https://doi.org/10.1109/SFCS.1983.5

Emmanuel Hainry, Romain Péchoux, and Mário Silva. 2023. A Programming Language Characterizing Quantum Polynomial Time. In *FoSSaCS 2023 (Lecture Notes in Computer Science, Vol. 13992)*. Springer, 156–175. https://doi.org/10.1007/978-3-031-30829-1_8

Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. 2005. *Skolem's Problem - On the Border between Decidability and Undecidability*. Technical Report 683.

Martin Hyland and Valeria de Paiva. 1993. Full Intuitionistic Linear Logic (extended abstract). *Annals of Pure and Applied Logic* 64, 3 (1993), 273–291. https://doi.org/10.1016/0168-0072(93)90146-5

Neil Immerman. 1999. *Descriptive complexity*. Springer, New York, NY, USA. https://doi.org/10.1007/978-1-4612-0539-5

Julia Kempe. 2003. Quantum random walks: An introductory overview. *Contemporary Physics* 44, 4 (jul 2003), 307–327. https://doi.org/10.1080/0010715031000110776

Emmanuel Knill. 1996. *Conventions for quantum pseudocode*. Technical Report LA-UR 96-2724. Los Alamos National Laboratory. http://arxiv.org/abs/2211.02559

Harry G. Mairson. 2004. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming* 14, 6 (2004), 623–633. https://doi.org/10.1017/S0956796804005131

Alexandre Miquel. 2011. A survey of classical realizability. In *TLCA 2011 (Lecture Notes in Computer Science, Vol. 6690)*. Springer, 1–2. https://doi.org/10.1007/978-3-642-21691-6_1

Piergiorgio Odifreddi. 1992. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, Amsterdam, The Netherlands.

Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: a core language for quantum circuits. In *POPL 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 846–858. https://doi.org/10.1145/3009837.3009894

Romain Péchoux. 2020. *Implicit computational complexity: Past and Future*. Habilitation à diriger des recherches. Université de Lorraine. https://hal.univ-lorraine.fr/tel-02978986

Lorenzo M. Procopio, Amir Moqanaki, Mateus Araújo, Fabio Costa, Irati Alonso Calafell, Emma G. Dowd, Deny R. Hamel, Lee A. Rozema, Časlav Brukner, and Philip Walther. 2015. Experimental superposition of orders of quantum gates. *Nature communications* 6 (2015), 7913. https://doi.org/10.1038/ncomms8913

John C. Reynolds. 1974. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974 (Lecture Notes in Computer Science, Vol. 19)*, Bernard J. Robinet (Ed.). Springer, 408–423. https://doi.org/10.1007/3-540-06859-7_148

Giulia Rubino, Lee A. Rozema, Adrien Feix, Mateus Araújo, Jonas M. Zeuner, Lorenzo M. Procopio, Časlav Brukner, and Philip Walther. 2017. Experimental verification of an indefinite causal order. *Science advances* 3, 3 (2017), e1602589. https://doi.org/10.1126/sciadv.1602589

Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. https://doi.org/10.1017/S0960129504004256

Peter Selinger and Benoît Valiron. 2006. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science* 16, 3 (2006), 527–552. https://doi.org/10.1017/S0960129506005238

Jaap Van Oosten (Ed.). 2008. *Realizability: an introduction to its categorical side.* Elsevier, Amsterdam, The Netherlands.

Finn Voichick, Liyi Li, Robert Rand, and Michael Hicks. 2023. Qunity: A Unified Language for Quantum and Classical Computing. *Proceedings of the ACM on Programming Languages* 7, POPL (2023), 921–951. https://doi.org/10.1145/3571225

William K. Wooters and Wojciech H. Zurek. 1982. A Single Quantum Cannot be Cloned. *Nature* 299 (1982), 802–803. https://doi.org/10.1038/299802a0

Tomoyuki Yamakami. 2020. A Schematic Definition of Quantum Polynomial Time Computability. *The Journal of Symbolic Logic* 85, 4 (2020), 1546–1587. https://doi.org/10.1017/jsl.2020.45

Andrew Chi-Chih Yao. 1993. Quantum Circuit Complexity. In *FOCS 1993*. IEEE Computer Society, 352–361. https://doi.org/10.1109/SFCS.1993.366852

Mingsheng Ying. 2016. *Foundations of quantum programming.* Morgan Kaufmann, Burlington, MA, USA.

Charles Yuan, Agnes Villanyi, and Michael Carbin. 2023. Quantum Control Machine: The Limits of Quantum Programs as Data. Preprint at arXiv. http://arxiv.org/abs/2304.15000

A PROOFS

A.1 Proofs of Section 3

Lemma A.1 (Substitution lemma). *For any two contexts Γ, Δ, the following hold:*

(1) If $\Gamma, x : A; \Delta \vdash \vec{t} : B$ and $; [z : C] \vdash \vec{u} : A$ then $[z : C], \Gamma; \Delta \vdash \vec{t}[\vec{u}/x] : B$.
(2) If $\Gamma; \Delta_1, x : A \vdash \vec{t} : B$ and $; \Delta_2 \vdash \vec{u} : A$ then $\Gamma; \Delta_1, \Delta_2 \vdash \vec{t}[\vec{u}/x] : B$.

Proof. By structural induction on t.
For 1, consider:

- If $\vec{t} = x$, then by a generation lemma we have that $\Delta = \emptyset$ and for some type $\vec{t}$ we have that $A = \S^{a_\S} T$ and $\sharp^b \S \{\sharp, \S\}^{a_\sharp, a_\S} T \leq B$, for $a_\sharp, a_\S, b \geqslant 0$. By another generation lemma we have that if $; [z : C] \vdash u : \S^{a_\S} T$ then we also have $; [z : C] \vdash u : \{\sharp, \S\}^{a_\sharp, a_\S} T$ and by $\S_i$ and $\sharp_i$ and finally by $\leq$ we obtain $[z; C]; \vdash u : B$, after which context Γ can be added via W.
- Let $\vec{t} = \lambda y.\vec{s}$. Then we consider two possible cases:
  - $\Gamma, x : A; y : C, \Delta \vdash \vec{s} : D$ and $C \multimap D \leq B$. By induction hypothesis we have that $[z : C], \Gamma; y : C, \Delta \vdash \vec{s}[u/x] : D$ and by rule $\multimap_i$ we obtain $[z : C], \Gamma; \Delta \vdash \lambda y.\vec{s}[u/x] : C \multimap D$ as intended. Notice in this case that $\vec{t}[y/x] = (\lambda y.\vec{s})[u/x] = \lambda y.(\vec{s}[u/x])$.
  - $\Gamma, x : A, y : C; \Delta \vdash \vec{s} : D$ and $C \Rightarrow D \leq B$. Similarly to the first case, by applying the induction hypothesis and rule $\Rightarrow_i$ we obtain $[z : C], \Gamma; \Delta \vdash \lambda y.\vec{s}[u/x] : C \Rightarrow D$.
- $\vec{t} = s\ r$. Let us consider two cases:
  - $\Gamma_1; \Delta_1 \vdash s : C \multimap D$ and $\Gamma_2; \Delta_2 \vdash r : C$, with $D \leq B$, where the contexts are such that $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$ and $(\Delta_1, \Delta_2) = \Delta$. Let us take the case where $\Gamma_1 = (x : A, \Gamma_1')$. Then by induction hypothesis we have that $\Gamma_1'; \Delta_1 \vdash s[u/x] : C \multimap D$. By rule $\multimap_i$ we can obtain $\Gamma_1', \Gamma_2; \Delta_1, \Delta_2 \vdash s[u/x]\vec{r} : D$, as desired. Notice that $(s\vec{r})[u/x] = s[u/x]\vec{r}$ in this case. The case where $x : A \in \Gamma_2$ is analogous.
  - $\Gamma_1; \Delta_1 \vdash s : C \Rightarrow D$ and $\Gamma_2; \Delta_2 \vdash r : C$, with $D \leq B$, where the contexts are such that $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$ and $(\Delta_1, \Delta_2) = \Delta$. This can be treated similarly to the previous case.
- Let $\vec{t} = \text{if } s \text{ then } \vec{p}_1 \text{ else } \vec{p}_2$. Consider two cases:
  - $\Gamma_1; \Delta_1 \vdash s : \mathbb{B}$ and $\Gamma_2; \Delta_2 \vdash (\vec{p}_1 \perp \vec{p}_2) : C$ such that $C \leq B$, with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$. If $x : A \in \Gamma_1$ then applying the induction hypothesis on s gives the desired result. If $\Gamma_2 = (x : A, \Gamma_2')$, then by induction hypothesis, we have that $\Gamma_2'; \Delta_2 \vdash (\vec{p}[u/x] \perp \vec{p}[u/x]) : C$. Notice that $\Gamma_2'; \Delta_2 \vdash \vec{p}[u/x] \perp \vec{p}[u/x]$ by definition of orthogonality for open superpositions: that they remain orthogonal for *any* substitution of open variables by closed superpositions. In particular this means that they remain orthogonal in partial substitutions by open superpositions, since if not we could find a total substitution that would be non-orthogonal. We can apply rule if and obtain $\Gamma_1, \Gamma_2'; \Delta_1, \Delta_2 \vdash \text{if } s \text{ then } \vec{p}_1[u/x] \text{ else } \vec{p}_2[u/x]$ as intended.
  - Case where $\Gamma_1; \Delta_1 \vdash s : \sharp\mathbb{B}$ is analogous.
- Let $\vec{t} = (t_1, t_2)$, in which case $\Gamma_1; \Delta_1 \vdash t_1 : B_1$ and $\Gamma_2; \Delta_2 \vdash t_2 : B_2$ with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$ and $(\Delta_1, \Delta_2) = \Delta$ and $B_1 \times B_2 \leq B$. Assume that $\Gamma_1 = (\Gamma_1', x : A)$, then, by induction hypothesis, $\Gamma_1'; \Delta_1 \vdash t_1[u/x] : B_1$ and so by rule $\times_i$ we have that $\Gamma_1', \Gamma_2; \Delta \vdash (t_1[u/x], t_2) : B$, which is our desired result. The case where $\Gamma_2 = (\Gamma_2', x : A)$ is analogous.
- If $\vec{t} = (\text{let } (x_1, x_2) = r \text{ in } \vec{s})$, then consider two options:
  - $\Gamma_1; \Delta_1 \vdash r : C \times D$ and $\Gamma_2, x_1 : C, x_2 : D; \Delta_2 \vdash \vec{s} : E$ such that $E \leq B$, with $(\Gamma_1, \Gamma_2) = (\Gamma, x : A)$ and $(\Delta_1, \Delta_2) = \Delta$. Assume that $\Gamma_1 = (\Gamma_1', x : A)$. Then, by induction hypothesis, $\Gamma_1'; \Delta_1 \vdash r[u/x] : C \times D$ and so by rule $\times_{e_1}$ we have that $\Gamma_1', \Gamma_2; \Delta \vdash \text{let } (x_1, x_2) = r[u/x] \text{ in } \vec{s} : E$. Notice that in this case $(\Gamma_1', \Gamma_2) = \Gamma$ and that $\text{let } (x_1, x_2) = r[u/x] \text{ in } \vec{s}$ is $(\text{let } (x_1, x_2) = r \text{ in } \vec{s})[u/x]$. The case where $\Gamma_2 = (\Gamma_2', x : A)$ is analogous, and so is the case for rule $\times_{e_2}$.

– $\Gamma_1; \Delta_1 \vdash r : \sharp(C \times D)$ and $\Gamma_2, x_1 : \sharp C, x_2 : \sharp D; \Delta_2 \vdash \vec{s} : E$ such that $\sharp E \leq B$. This case can also be treated via the induction hypothesis.

For 2, consider:

- If $\vec{t} = x$, by a generation lemma we have that $\Delta_1 = \emptyset$ and that for some type $\vec{t}$ we have $A = \S^{a_\S} T$ and $B = \{\sharp, \S\}^{a_\sharp, a_\S} T$. From $; \Delta_2 \vdash u : \S^{a_\S} T$ we can also derive $; \Delta_2 \vdash u : B$ by the fact that $\sharp_i$ does not alter the context. By W we obtain $\Gamma; \Delta_2 \vdash u : B$ as desired.
- Let $\vec{t} = \lambda y.\vec{s}$. Then we consider two possible cases:
  – $\Gamma; x : A, y : C, \Delta_1 \vdash \vec{s} : D$ and $C \multimap D \leq B$. By induction hypothesis we have that $\Gamma; \Delta_1, y : C, \Delta_2 \vdash \vec{s}[u/x] : D$ and by rule $\multimap_i$ we obtain $\Gamma; \Delta_1, \Delta_2 \vdash \lambda y.\vec{s}[u/x] : C \multimap D$ as intended. Notice in this case that $\vec{t}[y/x] = (\lambda y.\vec{s})[u/x] = \lambda y.(\vec{s}[u/x])$.
  – $\Gamma, y : C; x : A, \Delta_1 \vdash \vec{s} : D$ and $C \Rightarrow D \leq B$. Similarly to the first case, by applying the induction hypothesis and rule $\Rightarrow_i$ we obtain $\Gamma; \Delta_1, \Delta_2 \vdash \lambda y.\vec{s}[u/x] : C \Rightarrow D$.
- $\vec{t} = s\, r$. Let us consider two cases:
  – $\Gamma_1; \Delta_{1,1} \vdash s : C \multimap D$ and $\Gamma_2; \Delta_{1,2} \vdash r : C$, with $D \leq B$, where the contexts are such that $(\Gamma_1, \Gamma_2) = \Gamma$ and $(\Delta_{1,1}, \Delta_{1,2}) = (\Delta_1, x : A)$. Let us take the case where $\Delta_{1,1} = (x : A, \Delta'_{1,1})$. Then by induction hypothesis we have that $\Gamma_1; \Delta'_{1,1} \vdash s[u/x] : C \multimap D$. By rule $\multimap_i$ we can obtain $\Gamma_1, \Gamma_2; \Delta'_{1,1}, \Delta_{1,2} \vdash s[u/x]\vec{r} : D$, as desired. Notice that $(s\vec{r})[u/x] = s[u/x]\vec{r}$ in this case. The case where $x : A \in \Delta_{1,2}$ is analogous.
  – $\Gamma_1; \Delta_{1,1} \vdash s : C \Rightarrow D$ and $\Gamma_2; \Delta_2 \vdash r : C$, with $D \leq B$, can be treated similarly to the previous case.
- Let $\vec{t} = \mathsf{if}\ s\ \mathsf{then}\ \vec{p}_1\ \mathsf{else}\ \vec{p}_2$. Consider two cases:
  – $\Gamma_1; \Delta_{1,1} \vdash s : \mathbb{B}$ and $\Gamma_2; \Delta_{1,2} \vdash (\vec{p}_1 \bot \vec{p}_2) : C$ such that $C \leq B$, with $(\Gamma_1, \Gamma_2) = \Gamma$. If $x : A \in \Delta_{1,1}$ then applying the induction hypothesis on s gives the desired result. If $\Delta_{1,2} = (x : A, \Delta'_{1,2})$, then by induction hypothesis, we have that $\Gamma_2; \Delta'_{1,2} \vdash (\vec{p}[u/x] \bot \vec{p}[u/x]) : C$. We can apply rule if and obtain $\Gamma_1, \Gamma'_2; \Delta_1, \Delta_2 \vdash \mathsf{if}\ s\ \mathsf{then}\ \vec{p}_1[u/x]\ \mathsf{else}\ \vec{p}_2[u/x]$ as intended.
  – Case where $\Gamma_1; \Delta_{1,1} \vdash s : \sharp\mathbb{B}$ is analogous.
- Let $\vec{t} = (t_1, t_2)$, in which case $\Gamma_1; \Delta_{1,1} \vdash t_1 : B_1$ and $\Gamma_2; \Delta_2 \vdash t_2 : B_2$ with $(\Gamma_1, \Gamma_2) = \Gamma$ and $(\Delta_{1,1}, \Delta_{1,2}) = (\Delta_1, x : A)$ and $B_1 \times B_2 \leq B$. Assume that $\Delta_{1,1} = (\Delta'_{1,1}, x : A)$, then, by induction hypothesis, $\Gamma_1; \Delta'_{1,1} \vdash t_1[u/x] : B_1$ and so by rule $\times_i$ we have that $\Gamma; \Delta'_{1,1}, \Delta_{1,2} \vdash (t_1[u/x], t_2) : B$, which is our desired result. The case where $\Delta_{1,2} = (\Delta'_{1,2}, x : A)$ is analogous.
- If $\vec{t} = (\mathsf{let}\ \ (x_1, x_2) = r\ \mathsf{in}\ \vec{s})$, then consider two options:
  – $\Gamma_1; \Delta_{1,1} \vdash r : C \times D$ and $\Gamma_2, x_1 : C, x_2 : D; \Delta_{1,2} \vdash \vec{s} : E$ such that $E \leq B$, with $(\Gamma_1, \Gamma_2) = \Gamma$ and $(\Delta_{1,1}, \Delta_{1,2}) = \Delta_1$. Assume that $\Delta_{1,1} = (\Delta'_{1,1}, x : A)$. Then, by induction hypothesis, $\Gamma_1; \Delta'_{1,1} \vdash r[u/x] : C \times D$ and so by rule $\times_{e_1}$ we have that $\Gamma; \Delta'_{1,1}, \Delta_{1,2} \vdash \mathsf{let}\ \ (x_1, x_2) = r[u/x]\ \mathsf{in}\ \vec{s} : E$. Notice that in this case $\mathsf{let}\ \ (x_1, x_2) = r[u/x]\ \mathsf{in}\ \vec{s}$ is $(\mathsf{let}\ \ (x_1, x_2) = r\ \mathsf{in}\ \vec{s})[u/x]$. The case where $\Delta_{1,2} = (\Delta'_{1,2}, x : A)$ is analogous, and so is the case for rule $\times_{e_2}$.
  – $\Gamma_1; \Delta_1 \vdash r : \sharp(C \times D)$ and $\Gamma_2, x_1 : \sharp C, x_2 : \sharp D; \Delta_2 \vdash \vec{s} : E$ such that $\sharp E \leq B$. This case can also be treated via the induction hypothesis.

□

Theorem 3.3 (Subject reduction). *If* $; \vdash \vec{t} : A$ *and* $\vec{t} \rightsquigarrow \vec{r}$, *then* $; \vdash \vec{r} : A$.

Proof. By induction on the relation $\rightsquigarrow$.

- $\vec{t}$ is a value. Then, since $\vec{t}$ does not reduce, there is no $\vec{r}$ satisfying the condition and the theorem is trivially satisfied. In the next items assume v is a value, if not then apply induction hypothesis to v.
- $\vec{t} = (\lambda x.\vec{p})\ v$ and therefore $\vec{r} = \vec{p}[v/x]$. Then, from $; \vdash \vec{t} : A$ we consider two cases:

- $;\vdash \lambda x.\vec{p} : B \Rightarrow A$, with $x : B; \vdash \vec{p} : A$ and $;\vdash v : B$. Then by the substitution lemma we have that $;\vdash \vec{p}[v/x] : A$ which is the desired conclusion.
- $;\vdash \lambda x.\vec{p} : B \multimap A$, with $;x : B \vdash \vec{p} : A$ and $;\vdash v : B$. Substitution lemma provides the desired result.

- $\vec{t} = \mathsf{if}\ v\ \mathsf{then}\ \vec{p}\ \mathsf{else}\ \vec{s} : A$.
  If $;\vdash \vec{t}$ then rule if implies that $\vec{p} \perp_A \vec{s}$. There are two cases:
  (1) $v = |0\rangle$, and therefore $t \rightsquigarrow \vec{p}$ and the conclusion follows,
  (2) $v = |1\rangle$, and $t \rightsquigarrow \vec{s}$ and the conclusion also follows.
  Analogous case where we used rule $\mathsf{if}_\sharp$.
- $\vec{t} = \mathsf{let}\ (x, y) = (v, w)\ \mathsf{in}\ \vec{s}$, then consider 3 cases:
  (1) We used $\times_{e_1}$ and so from $;\vdash \vec{t} : A$ we have that $;\vdash (v, w) : B \times C$ and $;x : B, y : C \vdash \vec{s} : A$. This means that $;\vdash v : B$ and $;\vdash w : C$ and therefore by applying the substitution lemma twice we obtain $;\vdash \vec{s}[v/x, w/y] : A$ as desired.
  (2) Similar case to $\times_{e_2}$.
  (3) Similar case for $\times_{e_\sharp}$.

□

Theorem 3.4 (Progress). *If $;\vdash \vec{t} : A$, then either $\vec{t} \rightsquigarrow \vec{r}$ for some $\vec{r}$ or $\vec{t} \in \mathsf{V}$.*

Proof. By induction on the structure of $\vec{t}$.

- $\vec{t} = x$. Cannot be typed with a closed context.
- $\vec{t} = |0\rangle$. This is a value.
- $\vec{t} = |1\rangle$. This is a value.
- $\vec{t} = \mathsf{if}\ r\ \mathsf{then}\ \vec{s}_0\ \mathsf{else}\ \vec{s}_1$. Since r must be a boolean, either $r \rightsquigarrow \vec{r}'$ and therefore $\vec{t} \rightsquigarrow \mathsf{if}\ \vec{r}'\ \mathsf{then}\ \vec{s}_0\ \mathsf{else}\ \vec{s}_1$ by rule (If_+), or by induction hypothesis r must be a value and therefore either $r = |0\rangle$ or $r = |1\rangle$. In any of these two cases, $\vec{t}$ reduces, either by (If_0) or (If_1).
- $\vec{t} = \lambda x.\vec{p}$ is a value.
- $\vec{t} = s\ r$. We consider the three cases.
  (1) r is not a value. By the induction hypothesis, $r \rightsquigarrow \vec{r}'$ and by rule (App), $\vec{t} \rightsquigarrow s\ \vec{r}'$.
  (2) r is a value and s is not. By the induction hypothesis, $s \rightsquigarrow \vec{s}'$ and by rule (App_V), $\vec{t} \rightsquigarrow \vec{s}'\ r$.
  (3) Both r and s are values. Therefore $s = \lambda x.\vec{p}$ for some $\vec{p}$ and by rule (Abs) we have that $\vec{t} = (\lambda x.\vec{p})\ r \rightsquigarrow \vec{p}[s/x]$.
- $\vec{t} = (s, r)$. We consider the two possible cases where $\vec{t}$ is not a value:
  (1) s is a value and r is not, in which case by the induction hypothesis $r \rightsquigarrow \vec{r}'$ and therefore by rule (Pair_V) we have that $\vec{t} \rightsquigarrow (s, \vec{r}')$.
  (2) s is not a value, in which case by the induction hypothesis $s \rightsquigarrow \vec{s}'$ and therefore by rule (Pair) we have that $\vec{t} \rightsquigarrow (\vec{s}', r)$.
- $\vec{t} = \mathsf{let}\ (x, y) = s\ \mathsf{in}\ \vec{r}$. Consider two cases:
  (1) s is a value of the form (v, w), in which case by rule (Let), we have $\vec{t} \rightsquigarrow \vec{r}[v/x, w/y]$.
  (2) s is not a value and therefore by the induction hypothesis $s \rightsquigarrow \vec{s}'$ and we have that $\vec{t} \rightsquigarrow \mathsf{let}\ (x, y) = \vec{s}'\ \mathsf{in}\ \vec{r}$.
- $\vec{t} = \vec{0}$. This is a value.
- $\vec{t} = \alpha \cdot \vec{s}$, in which case either $\vec{s}$ is a value and therefore $\vec{t}$ is also a value, or $\vec{s} \rightsquigarrow \vec{s}'$ and by rule (Sup), $\vec{t} \rightsquigarrow \alpha \cdot \vec{s}'$.
- $\vec{t} = \vec{s} + \vec{r}$, and by rule (Sup) if $\vec{s}$ or $\vec{r}$ are not values then $\vec{t}$ also reduces.

□

A.2 Proofs of Section 5

Lemma 5.1. *Let $A \in \mathbb{T}_s$, then $[\![!(A)]\!]_\emptyset = \flat[\![A]\!]_\emptyset$.*

Proof. First we state the following straightforward properties.

(1) For any set $S \subseteq \mathsf{BV}_c$, we have $S = \flat S$.
(2) For any set $S \subseteq \mathsf{V}_c$, we have $\flat S = \flat\,\mathrm{span}(S)$.
(3) For any set $S \subseteq \mathsf{BV}_c$, we have $S = S \cap \mathcal{S}_1$.

Then we proceed by induction on the structure of simple types.

- If $A = \mathbb{B}$, then $!(A) = !(\mathbb{B}) = \mathbb{B}$. Hence
$$[\![!(A)]\!]_\emptyset = [\![\mathbb{B}]\!]_\emptyset =^{(1)} \flat[\![\mathbb{B}]\!]_\emptyset = \flat[\![A]\!]_\emptyset$$
- If $A = B \multimap C$, then $!(A) = !(B \multimap C) = B \multimap C$. Hence,
$$[\![!(A)]\!]_\emptyset = [\![!(B \multimap C)]\!]_\emptyset = [\![B \multimap C]\!]_\emptyset =^{(1)} \flat[\![B \multimap C]\!]_\emptyset = \flat[\![A]\!]_\emptyset$$
- If $A = B \Rightarrow C$, then $!(A) = !(B \Rightarrow C) = B \Rightarrow C$. Hence,
$$[\![!(A)]\!]_\emptyset = [\![!(B \Rightarrow C)]\!]_\emptyset = [\![B \Rightarrow C]\!]_\emptyset =^{(1)} \flat[\![B \Rightarrow C]\!]_\emptyset = \flat[\![A]\!]_\emptyset$$
- If $A = B \times C$, then $!(A) = !(B) \times !(C)$. Hence,
$$\begin{aligned}[\![!(A)]\!]_\emptyset &= [\![!(B) \times !(C)]\!]_\emptyset = \{(\vec{v}, \vec{w}) : \vec{v} \in [\![!(B)]\!]_\emptyset, \vec{w} \in [\![!(C)]\!]_\emptyset\} \\ &=^{(\mathrm{IH})} \{(\vec{v}, \vec{w}) : \vec{v} \in \flat[\![B]\!]_\emptyset, \vec{w} \in \flat[\![C]\!]_\emptyset\} = \flat\{(\vec{v}, \vec{w}) : \vec{v} \in [\![B]\!]_\emptyset, \vec{w} \in [\![C]\!]_\emptyset\} \\ &= \flat([\![B \times C]\!]_\emptyset) = \flat[\![A]\!]_\emptyset\end{aligned}$$
- If $A = \sharp B$, then $!(A) = !(\sharp B) = !(B)$. Hence,
$$\begin{aligned}[\![!(A)]\!]_\emptyset = [\![!(B)]\!]_\emptyset &=^{(\mathrm{IH})} \flat[\![B]\!]_\emptyset =^{(2)} \flat\mathrm{span}([\![B]\!]_\emptyset) =^{(3)} \flat\mathrm{span}([\![B]\!]_\emptyset) \cap \mathcal{S}_1 \\ &=^{(3)} \flat(\mathrm{span}([\![B]\!]_\emptyset) \cap \mathcal{S}_1) = \flat[\![\sharp B]\!]_\emptyset = \flat[\![A]\!]_\emptyset\end{aligned}$$
- If $A = \S B$, then $!(A) = !(\S B) = \S !(B)$. Hence,
$$[\![!(A)]\!]_\emptyset = [\![\S !(B)]\!]_\emptyset = [\![!(B)]\!]_\emptyset =^{(\mathrm{IH})} \flat[\![B]\!]_\emptyset = \flat[\![\S B]\!]_\emptyset = \flat[\![A]\!]_\emptyset$$

□

Lemma 5.2. *Let $A, B \in \mathbb{T}$ and τ defined in $FV(A, B)$. Then, $[\![A[B/X]]\!]_\tau = [\![A]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}$.*

Proof. By induction on A.

- If $A = X$, then $A[B/X] = B$. Hence,
$$[\![A[B/X]]\!]_\tau = [\![B]\!]_\tau = [\![X]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}} = [\![A]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}$$
- If $A = Y \neq X$, then $A[B/X] = Y$. Hence,
$$[\![A[B/X]]\!]_\tau = [\![Y]\!]_\tau = \tau(Y) = (\tau \cup \{X \mapsto [\![B]\!]_\tau\})(Y) = [\![Y]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}} = [\![A]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}$$
- If $A = \mathbb{B}$, then $A[B/X] = \mathbb{B}$. Hence,
$$[\![A[B/X]]\!]_\tau = [\![\mathbb{B}]\!]_\tau = \{|0\rangle, |1\rangle\} = [\![\mathbb{B}]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}} = [\![A]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}$$
- If $A = C \multimap D$, then $A[B/X] = C[B/X] \multimap D[B/X]$. Hence,
$$\begin{aligned}[\![A[B/X]]\!]_\tau &= [\![C[B/X] \multimap D[B/X]]\!]_\tau = \{\lambda x.\vec{t} : \vec{v} \in [\![C[B/X]]\!]_\tau, \vec{t}\langle\vec{v}/x\rangle \Vdash [\![D[B/X]]\!]_\tau\} \\ &=^{(\mathrm{IH})} \{\lambda x.\vec{t} : \vec{v} \in [\![C]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}, \vec{t}\langle\vec{v}/x\rangle \Vdash [\![D]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}\} \\ &= [\![C \multimap D]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}} = [\![A]\!]_{\tau \cup \{X \mapsto [\![B]\!]_\tau\}}\end{aligned}$$

- If $A = C \Rightarrow D$, then $A[B/X] = C[B/X] \Rightarrow D[B/X]$. Hence,

$$\begin{aligned}\llbracket A[B/X] \rrbracket_\tau &= \llbracket C[B/X] \Rightarrow D[B/X] \rrbracket_\tau = \{\lambda x.\vec{t} : \vec{v} \in \flat \llbracket C[B/X] \rrbracket_\tau, \vec{t}[\vec{v}/x] \Vdash \llbracket D[B/X] \rrbracket_\tau\} \\ &=^{(\mathrm{IH})} \{\lambda x.\vec{t} : \vec{v} \in \flat \llbracket C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}, \vec{t}[\vec{v}/x] \Vdash \llbracket D \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\} \\ &= \llbracket C \Rightarrow D \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket A \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\end{aligned}$$

- If $A = C \times D$, then $A[B/X] = C[B/X] \times D[B/X]$. Hence,

$$\begin{aligned}\llbracket A[B/X] \rrbracket_\tau &= \llbracket C[B/X] \times D[B/X] \rrbracket_\tau = \{(\vec{v}, \vec{w}) : \vec{v} \in \llbracket C[B/X] \rrbracket_\tau, \vec{w} \in \llbracket D[B/X] \rrbracket_\tau\} \\ &=^{(\mathrm{IH})} \{(\vec{v}, \vec{w}) : \vec{v} \in \llbracket C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}, \vec{w} \in \llbracket D \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}, \} \\ &= \llbracket C \times D \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket A \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\end{aligned}$$

- If $A = \sharp C$, then $A[B/X] = \sharp C[B/X]$. Hence,

$$\begin{aligned}\llbracket A[B/X] \rrbracket_\tau &= \llbracket \sharp C[B/X] \rrbracket_\tau = \mathsf{span}(\llbracket C[B/X] \rrbracket_\tau) \cap \mathcal{S}_1 \\ &=^{(\mathrm{IH})} \mathsf{span}(\llbracket C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}) \cap \mathcal{S}_1 = \llbracket \sharp C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket A \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\end{aligned}$$

- If $A = \S C$, then $A[B/X] = \S C[B/X]$. Hence,

$$\begin{aligned}\llbracket A[B/X] \rrbracket_\tau &= \llbracket \S C[B/X] \rrbracket_\tau = \llbracket C[B/X] \rrbracket_\tau \\ &=^{(\mathrm{IH})} \llbracket C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket \S C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket A \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\end{aligned}$$

- If $A = \forall Y.C$, then $A[B/X] = \forall Y.C[B/X]$. Hence,

$$\begin{aligned}\llbracket A[B/X] \rrbracket_\tau &= \llbracket \forall Y.C[B/X] \rrbracket_\tau = \bigcap_{R \subseteq \mathcal{S}_1} \llbracket C[B/X] \rrbracket_{\tau \cup \{Y \mapsto R\}} \\ &=^{(\mathrm{IH})} \bigcap_{R \subseteq \mathcal{S}_1} \llbracket C \rrbracket_{\tau \cup \{Y \mapsto R, X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket \forall Y.C \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}} = \llbracket A \rrbracket_{\tau \cup \{X \mapsto \llbracket B \rrbracket_\tau\}}\end{aligned}$$

□

Theorem 5.3. *The typing rules in Figure 6 are valid.*

Proof.

(W) Assume that $\Gamma; \Delta \vdash \vec{t} : A$ is a valid typing judgment. Then $\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma, \Delta)$, and $\vec{t}\langle\sigma\rangle \Vdash A$, for any $\sigma \in \llbracket \Gamma, \Delta \rrbracket_\tau$. Given a variable x with type B such that $x \notin \mathrm{dom}(\Gamma, \Delta)$, we have that $\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma, \Delta) \cup \{x\}$. Now, for any substitution $\sigma \in \llbracket \Gamma, \Delta, \{x : B\} \rrbracket_\tau$, we have that $\sigma = \sigma_0\langle\vec{v}/x\rangle$ for some $\sigma_0 \in \llbracket \Gamma, \Delta \rrbracket_\tau$ and $\vec{v} \Vdash B$. Then,

$$\vec{t}\langle\sigma\rangle = \vec{t}[\vec{v}/x]\langle\sigma_0\rangle = \vec{t}\langle\sigma_0\rangle \Vdash A,$$

since $\sigma_0 \in \llbracket \Gamma, \Delta \rrbracket_\tau$ and $x \notin FV(\vec{t})$.

(C) The premise being valid, this means that $\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma, \Delta) \cup \{x, y\}$ and for all $\sigma \in \llbracket \Gamma, \Delta, \{x : B, y : B\} \rrbracket_\tau$ we have that $\vec{t}\langle\sigma\rangle \Vdash A$. It is clear that $FV(\vec{t}[x/y]) \subseteq \mathrm{dom}(\Gamma, \Delta) \cup \{x\}$. For any $\sigma \in \llbracket \Gamma, \Delta, \{x : B\} \rrbracket_\tau$, $\sigma = \sigma_0\langle\vec{v}/x\rangle$ for some $\sigma_0 \in \llbracket \Gamma, \Delta \rrbracket_\tau$ and some value $\vec{v} \in \llbracket B \rrbracket_\tau$. Therefore,

$$\begin{aligned}(\vec{t}[x/y])\langle\sigma\rangle &= (\vec{t}[x/y])\langle x/\vec{v}, \sigma_0\rangle = (\vec{t}[x/y][\vec{v}/x])\langle\sigma_0\rangle \\ &= (\vec{t}[\vec{v}/x][x/y])\langle\sigma_0\rangle = (\vec{t}[\vec{v}/x][\vec{v}/y])\langle\sigma_0\rangle \\ &= \vec{t}\langle\vec{v}/x, \vec{v}/y, \sigma_0\rangle \Vdash A,\end{aligned}$$

since $\langle\vec{v}/x, \vec{v}/y, \sigma_0\rangle \in \llbracket \Gamma, \Delta, \{x : B, y : B\} \rrbracket_\tau$.

(≡) By associativity of the substitution.

(≤) By definition of subtyping.

(ax) We have that $\mathrm{dom}(\Delta) = \{x\} = FV(x)$ and $\mathrm{dom}(\Gamma) = \tau$. For any substitution $\sigma \in [\![x : A]\!]_\tau$, we have $\sigma = \{\vec{v}/x\}$ for some $\vec{v} \in [\![A]\!]_\tau$. Therefore $x\langle\sigma\rangle = x\langle\vec{v}/x\rangle = \vec{v} \Vdash [\![A]\!]_\tau$.

(0) The rule is valid since $FV(|0\rangle) = \emptyset$ and $|0\rangle \in [\![\mathbb{B}]\!]_\emptyset$ by definition.

(1) Similar to (0).

(if) Supposing that the premises are valid, we obtain:

(1) $\mathrm{dom}(\Delta_1) \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1)$ and $t\langle\sigma\rangle \Vdash [\![\mathbb{B}]\!]_\emptyset$ for all $\sigma \in [\![\Gamma_1, \Delta_1]\!]_\tau$.

(2) $\mathrm{dom}(\Delta_2) \subseteq FV(\vec{r}) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2)$ and $\vec{r}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

(3) $\mathrm{dom}(\Delta_2) \subseteq FV(\vec{s}) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2)$ and $\vec{s}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

From (1)–(3) we have that $\mathrm{dom}(\Delta_1, \Delta_2) \subseteq FV(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$. Given a $\sigma \in [\![\Gamma_1, \Delta_1, \Gamma_2, \Delta_2]\!]_\tau$, then $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i, \Delta_i]\!]_\tau$, and we have that

$$\begin{aligned}&(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma\rangle\\ =\ &(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma_1\rangle\langle\sigma_2\rangle\\ =\ &\text{if } t\langle\sigma_1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle.\end{aligned}$$

Since $\sigma_1 \in [\![\Gamma_1, \Delta_1]\!]_\tau$ we have that $t\langle\sigma_1\rangle \Vdash [\![\mathbb{B}]\!]_\emptyset$ from (1). Therefore we can consider two cases:

– $t\langle\sigma_1\rangle \rightsquigarrow^* |0\rangle$ such that

$$\begin{aligned}&(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma\rangle\\ =\ &\text{if } t\langle\sigma_1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle\\ \rightsquigarrow^*\ &\text{if } |0\rangle \text{then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle\\ \rightsquigarrow^*\ &\vec{r}\langle\sigma_2\rangle \Vdash [\![A]\!]_\tau\end{aligned}$$

using (2) and the fact that $\sigma_2 \in [\![\Gamma_2, \Delta_2]\!]_\tau$. The second case is similar:

– $t\langle\sigma_1\rangle \rightsquigarrow^* |1\rangle$ such that

$$\begin{aligned}&(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma\rangle\\ =\ &\text{if } t\langle\sigma_1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle\\ \rightsquigarrow^*\ &\text{if } |1\rangle \text{then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle\\ \rightsquigarrow^*\ &\vec{s}\langle\sigma_2\rangle \Vdash [\![A]\!]_\tau.\end{aligned}$$

(if$_\sharp$) From the premises we deduce:

(1) $\mathrm{dom}(\Delta_1) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![\sharp\mathbb{B}]\!]_\emptyset$ for all $\sigma \in [\![\Gamma_1, \Delta_1]\!]_\tau$.

(2) $\mathrm{dom}(\Delta_2) \subseteq FV(\vec{r}) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2)$ and $\vec{r}\langle\sigma\rangle \Vdash [\![Q]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

(3) $\mathrm{dom}(\Delta_2) \subseteq FV(\vec{s}) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2)$ and $\vec{s}\langle\sigma\rangle \Vdash [\![Q]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

(4) For all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$, we have that $\vec{r}\langle\sigma\rangle \rightsquigarrow^* \vec{v}$ and $\vec{s}\langle\sigma\rangle \rightsquigarrow^* \vec{w}$ such that $\langle\vec{v}, \vec{w}\rangle = 0$.

Similarly to (if), we have that $\mathrm{dom}(\Delta_1, \Delta_2) \subseteq FV(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$ and, given a $\sigma \in [\![\Gamma_1, \Delta_1, \Gamma_2, \Delta_2]\!]_\tau$, then $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i, \Delta_i]\!]_\tau$, and we have that

$$\begin{aligned}&(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma\rangle\\ =\ &(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma_1\rangle\langle\sigma_2\rangle\\ =\ &\text{if } t\langle\sigma_1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle.\end{aligned}$$

From (1) and the fact that $\sigma_1 \in [\![\Gamma_1, \Delta_1]\!]_\tau$ we deduce $t\langle\sigma_1\rangle \Vdash \sharp\mathbb{B}$ and therefore $t\langle\sigma_1\rangle \rightsquigarrow^* \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, for $\alpha, \beta \in \overline{\mathbb{Q}}$ such that $|\alpha|^2 + |\beta|^2 = 1$. Therefore,

$$\begin{aligned}
&(\text{if } t \text{ then } \vec{r} \text{ else } \vec{s})\langle\sigma\rangle \\
&= \text{if } t\langle\sigma_1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle \\
&\rightsquigarrow^* \text{if } (\alpha \cdot |0\rangle + \beta \cdot |1\rangle) \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle \\
&= \alpha \cdot (\text{if } |0\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle) + \beta \cdot (\text{if } |1\rangle \text{ then } \vec{r}\langle\sigma_2\rangle \text{ else } \vec{s}\langle\sigma_2\rangle) \\
&\rightsquigarrow^* \alpha \cdot \vec{r}\langle\sigma_2\rangle + \beta \cdot \vec{s}\langle\sigma_2\rangle \\
&\rightsquigarrow^* \alpha \cdot \vec{v} + \beta \cdot \vec{w} \Vdash [\![\sharp Q]\!]_\tau
\end{aligned}$$

derived from (2)–(4) and the fact that $\sigma_2 \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

($\multimap_i$) Suppose that $\Gamma; \Delta, x : A \vdash \vec{t} : B$ is a valid judgment. Then since $(\text{dom}(\Delta) \cup \{x\}) \subseteq FV(\vec{t})$ we have $\text{dom}(\Delta) \subseteq FV(\lambda x.\vec{t})$. Similarly, since $FV(\vec{t}) \subseteq (\text{dom}(\Gamma, \Delta) \cup \{x\})$ it is also true that $FV(\lambda x.\vec{t}) \subseteq \text{dom}(\Gamma, \Delta)$. For any $\sigma \in [\![\Gamma, \Delta]\!]_\tau$, we have that $(\lambda x.\vec{t})\langle\sigma\rangle = \lambda x.\vec{t}\langle\sigma\rangle$, since $x \notin \text{dom}(\sigma)$. For all $\vec{v} \in [\![A]\!]_\tau$, we observe that $(\vec{t}\langle\sigma\rangle)\langle\vec{v}/x\rangle = \vec{t}\langle\sigma, \{\vec{v}/x\}\rangle \Vdash [\![B]\!]_\tau$, since $\sigma, \{\vec{v}/x\} \in [\![\Gamma, x : A]\!]_\tau$. Therefore, $(\lambda x.\vec{t})\langle\sigma\rangle \Vdash [\![A \multimap B]\!]_\tau$.

($\Rightarrow_i$) Suppose that $\Gamma, x : A; \Delta \vdash t : B$ is a valid judgment. Then $\text{dom}(\Delta) \subseteq FV(t) \subseteq \text{dom}(\Gamma, \{x : A\}, \Delta)$. We can therefore conclude that $\text{dom}(\Delta) \subseteq FV(\lambda x.t) \subseteq \text{dom}(\Gamma, \Delta)$. Given $\sigma \in [\![\Gamma, \Delta]\!]_\tau$ we want to show that $(\lambda x.t)\langle\sigma\rangle \Vdash [\![!(A) \Rightarrow B]\!]_\tau$. Since $x \notin \text{dom}(\sigma)$, we have $(\lambda x.t)\langle\sigma\rangle = \lambda x.(t\langle\sigma\rangle)$, and for all values $\vec{v} \in [\![A]\!]_\tau$, we have that

$$t\langle\sigma\rangle\langle\vec{v}/x\rangle = t\langle\sigma, \{\vec{v}/x\}\rangle \Vdash [\![B]\!]_\tau$$

from the premise. Since $\flat[\![A]\!]_\tau \subseteq [\![A]\!]_\tau$, we have that $(\lambda x.t)\langle\sigma\rangle \Vdash [\![!(A) \Rightarrow B]\!]_\tau$.

($\multimap_e$) Suppose that both judgments $\Gamma_1; \Delta_1 \vdash t : A \multimap B$ and $\Gamma_2; \Delta_2 \vdash \vec{r} : A$ are valid, meaning that:
- $\text{dom}(\Delta_1) \subseteq FV(t) \subseteq \text{dom}(\Gamma_1, \Delta_1)$ and $t\langle\sigma\rangle \Vdash [\![A \multimap B]\!]_\tau$ for all $\sigma \in [\![\Gamma_1, \Delta_1]\!]_\tau$.
- $\text{dom}(\Delta_2) \subseteq FV(\vec{r}) \subseteq \text{dom}(\Gamma_2, \Delta_2)$ and $\vec{r}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

This implies that $\text{dom}(\Delta_1, \Delta_2) \subseteq FV(t\,\vec{r}) \subseteq \text{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$. Given $\sigma \in [\![\Gamma_1, \Delta_1, \Gamma_2, \Delta_2]\!]_\tau$,then $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i, \Delta_i]\!]_\tau$, and from $FV(t) \bigcap \text{dom}(\sigma_2) = FV(\vec{r}) \bigcap \text{dom}(\sigma_1) = \emptyset$, then

$$(t\,\vec{r})\langle\sigma\rangle = (t\,\vec{r})\langle\sigma_1\rangle\langle\sigma_2\rangle = (t\langle\sigma_1\rangle\,\vec{r})\langle\sigma_2\rangle = t\langle\sigma_1\rangle\,\vec{r}\langle\sigma_2\rangle,$$

and so we conclude that $(t\,\vec{r})\langle\sigma\rangle = t\langle\sigma_1\rangle\,\vec{r}\langle\sigma_2\rangle \Vdash [\![B]\!]_\tau$, by the application of realizers [Díaz-Caro et al. 2019b, Lemma A.4].

($\Rightarrow_e$) Assuming the premises are valid, we have that
- $\text{dom}(\Delta) \subseteq FV(t) \subseteq \text{dom}(\Gamma, \Delta)$ and $t\langle\sigma\rangle \Vdash [\![!(A) \Rightarrow B]\!]_\tau$ for all $\sigma \in [\![\Gamma, \Delta]\!]_\tau$.
- $FV(r) = \{z\}$ and $r\langle\sigma\rangle \Vdash !A$ for all $\sigma \in [\![z : C]\!]_\tau$.

We can directly conclude that $\text{dom}(\Delta) \subseteq FV(t) \subseteq \text{dom}(\Gamma, \Delta) \cup \{z\}$. From the fact that $z \notin \text{dom}(\Gamma, \Delta)$, we have that for any $\sigma \in [\![\Gamma, \Delta, \{z : C\}]\!]_\tau$, $\sigma = \sigma_1\sigma_2$ for $\sigma_1 \in [\![\Gamma, \Delta]\!]_\tau$ and $\sigma_2 \in [\![\{z : C\}]\!]_\tau$. Therefore

$$(t\,r)\langle\sigma\rangle = (t\,r)\langle\sigma_1\rangle\langle\sigma_2\rangle = t\langle\sigma_1\rangle\,r\langle\sigma_2\rangle \Vdash [\![B]\!]_\tau,$$

by the application of realizers.

($\times_i$) Assuming that $\Gamma_1; \Delta_1 \vdash \vec{t} : A$ and $\Gamma_2; \Delta_2 \vdash \vec{r} : B$ are valid typing judgments, we have that
(1) $\text{dom}(\Delta_1) \subseteq FV(\vec{t}) \subseteq \text{dom}(\Gamma_1, \Delta_1)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all $\sigma \in [\![\Gamma_1, \Delta_1]\!]_\tau$.
(2) $\text{dom}(\Delta_2) \subseteq FV(\vec{r}) \subseteq \text{dom}(\Gamma_2, \Delta_2)$ and $\vec{r}\langle\sigma\rangle \Vdash [\![B]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

We have that $\mathrm{dom}(\Delta_1,\Delta_2) \subseteq FV((\vec{t},\vec{r})) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1,\Gamma_2,\Delta_2)$. For any $\sigma \in [\![\Gamma_1,\Delta_1,\Gamma_2,\Delta_2]\!]_\tau$ we have that $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i,\Delta_i]\!]_\tau$ and, from the disjointness of contexts, we deduce:

$$\begin{aligned}(\vec{t},\vec{r})\,\langle\sigma\rangle &= (\vec{t},\vec{r})\,\langle\sigma_1\rangle\langle\sigma_2\rangle \\ &= (\vec{t}\langle\sigma_1\rangle,\vec{r}\langle\sigma_2\rangle) \Vdash [\![A\times B]\!]_\tau,\end{aligned}$$

derived from the definition of $[\![A\times B]\!]_\tau$.

($\times_e$) Supposing that the judgments $\Gamma_1;\Delta_1 \vdash t : A\times B$ and $\Gamma_2;\Delta_2, x:A, y:B \vdash \vec{s} : C$ are valid, we obtain:

(1) $\mathrm{dom}(\Delta_1) \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1)$ and $t\langle\sigma\rangle \Vdash [\![A\times B]\!]_\tau$ for all $\sigma \in [\![\Gamma_1,\Delta_1]\!]_\tau$.

(2) $\mathrm{dom}(\Delta_2)\cup\{x,y\} \subseteq FV(\vec{s}) \subseteq \mathrm{dom}(\Gamma_2,\Delta_2)\cup\{x,y\}$ and $\vec{s}\langle\sigma\rangle \Vdash [\![C]\!]_\tau$ for all $\sigma \in [\![\Gamma_2,\Delta_2,\{x:A, y:B\}]\!]_\tau$.

We have that $\mathrm{dom}(\Delta_1,\Delta_2) \subseteq FV(\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1,\Gamma_2,\Delta_2)$. For any $\sigma \in [\![\Gamma_1,\Delta_1,\Gamma_2,\Delta_2]\!]_\tau$ we have that $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i,\Delta_i]\!]_\tau$ and therefore

$$\begin{aligned}(\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s})\langle\sigma\rangle &= (\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s})\langle\sigma_1\rangle\langle\sigma_2\rangle \\ &= \mathsf{let}\ (x,y) = t\langle\sigma_1\rangle\ \mathsf{in}\ \vec{s}\langle\sigma_2\rangle \\ &\rightsquigarrow^* \mathsf{let}\ (x,y) = \langle\vec{v},\vec{w}\rangle\ \mathsf{in}\ \vec{s}\langle\sigma_2\rangle \\ &\rightsquigarrow^* (\vec{s}[\vec{v}/x,\vec{w}/y])\langle\sigma_2\rangle \\ &= \vec{s}\langle\vec{v}/x,\vec{w}/y,\sigma_2\rangle \Vdash [\![C]\!]_\tau\end{aligned}$$

from the fact that $\langle\vec{v}/x,\vec{w}/y,\sigma_2\rangle \in [\![\Gamma_2,\Delta_2,\{x:A,y:B\}]\!]_\tau$ and [Díaz-Caro et al. 2019b, Lemmas A.10 and A.3].

($\times_{e\sharp}$) From the validity of $\Gamma_1;\Delta_1 \vdash \vec{t} : \sharp(Q\times R)$ and $\Gamma_2;\Delta_2, x:\sharp Q, y:\sharp R \vdash \vec{s} : S$ we have that

(1) $\mathrm{dom}(\Delta_1) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![\sharp(Q\times R)]\!]_\tau$ for all $\sigma \in [\![\Gamma_1,\Delta_1]\!]_\tau$.

(2) $\mathrm{dom}(\Delta_2)\cup\{x,y\} \subseteq FV(\vec{s}) \subseteq \mathrm{dom}(\Gamma_2,\Delta_2)\cup\{x,y\}$ and $\vec{s}\langle\sigma\rangle \Vdash [\![S]\!]_\tau$ for all $\sigma \in [\![\Gamma_2,\Delta_2,\{x:\sharp Q, y:\sharp R\}]\!]_\tau$.

We obtain directly that $\mathrm{dom}(\Delta_1,\Delta_2) \subseteq FV(\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1,\Gamma_2,\Delta_2)$ and, for any $\sigma \in [\![\Gamma_1,\Delta_1,\Gamma_2,\Delta_2]\!]_\tau$ we have that $\sigma = \sigma_1\sigma_2$ for $\sigma_i \in [\![\Gamma_i,\Delta_i]\!]_\tau$ and thus we have:

$$\begin{aligned}(\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s})\langle\sigma\rangle &= (\mathsf{let}\ (x,y) = t\ \mathsf{in}\ \vec{s})\langle\sigma_1\rangle\langle\sigma_2\rangle \\ &= \mathsf{let}\ (x,y) = t\langle\sigma_1\rangle\ \mathsf{in}\ \vec{s}\langle\sigma_2\rangle \\ &\rightsquigarrow^* \mathsf{let}\ (x,y) = \sum_{i=1}^{n}\alpha_i\cdot(\vec{v}_i,\vec{w}_i)\ \mathsf{in}\ \vec{s}\langle\sigma_2\rangle \\ &= \sum_{i=1}^{n}\alpha_i\cdot\mathsf{let}\ (x,y) = (\vec{v}_i,\vec{w}_i)\ \mathsf{in}\ \vec{s}\langle\sigma_2\rangle \\ &\rightsquigarrow^* \sum_{i=1}^{n}\alpha_i\cdot\vec{s}[\vec{v}_i/x,\vec{w}_i/y]\langle\sigma_2\rangle \\ &= \sum_{i=1}^{n}\alpha_i\cdot\vec{s}\langle\vec{v}_i/x,\vec{w}_i/y,\sigma_2\rangle \rightsquigarrow^* \sum_{i=1}^{n}\alpha_i\cdot\vec{z}_i \in \mathrm{span}([\![S]\!]_\tau)\end{aligned}$$

To see that this has unit norm, consider:

$$\begin{aligned}\Big\|\sum_{i=1}^{n}\alpha_i\cdot\vec{z}_i\Big\|^2 &= \Big\langle\sum_{i=1}^{n}\alpha_i\cdot\vec{z}_i\;\Big|\;\sum_{i=1}^{n}\alpha_i\cdot\vec{z}_i\Big\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\bar{\alpha}_j\langle\vec{z}_i|\vec{z}_j\rangle\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\bar{\alpha}_j\langle\vec{v}_i|\vec{v}_j\rangle\langle\vec{w}_i|\vec{w}_j\rangle\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\bar{\alpha}_j\langle(\vec{v}_i,\vec{w}_i)\mid(\vec{v}_j,\vec{w}_j)\rangle\\ &= \langle\sum_{i=1}^{n}\alpha_i(\vec{v}_i,\vec{w}_i)\mid\sum_{i=1}^{n}\alpha_i(\vec{v}_i,\vec{w}_i)\rangle = \Big\|\sum_{i=1}^{n}\alpha_i\cdot(\vec{v}_i,\vec{w}_i)\Big\|^2 = 1,\end{aligned}$$

where we have used [Díaz-Caro et al. 2019b, Lemma A.9 and Proposition A.2].

($\S_i$) Assuming that the premise is a valid typing judgment, we have that $FV(\vec{t}) = \mathrm{dom}(\Gamma,\Delta)$. Using the fact that $[\![\S B]\!]_\tau \triangleq [\![B]\!]_\tau$, we directly deduce that $\mathrm{dom}(\S\Delta) = \mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma,\Delta) = \mathrm{dom}(\Gamma,\S\Delta)$. Since $[\![\Gamma,\Delta]\!]_\tau = [\![\Gamma,\S\Delta]\!]_\tau$ we conclude that the rule is valid.

($\sharp_i$) If the premises are valid then we have that $\mathrm{dom}(\Delta) \subseteq FV(\vec{t}_i) \subseteq \mathrm{dom}(\Gamma,\Delta)$, for all i. Furthermore, for all $\sigma \in [\![\Gamma,\Delta]\!]$, $\vec{t}_i\langle\sigma\rangle \Vdash [\![A]\!]_\tau$. Since $\sum_{i=1}^{n}|\alpha_i|^2 = 1$, we have that

$$\Big(\sum_{i=1}^{n}\alpha_i\cdot\vec{t}_i\Big)\langle\sigma\rangle = \sum_{i=1}^{n}\alpha_i\cdot(\vec{t}_i\langle\sigma\rangle) \Vdash [\![\sharp A]\!]_\tau.$$

($\S_e$) If the premises are valid then:
(1) $\mathrm{dom}(\Delta_1) \subseteq FV(\vec{r}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1)$ and for all $\sigma \in [\![\Gamma_1,\Delta_1]\!]_\tau$, $\vec{r}\langle\sigma\rangle \Vdash [\![B]\!]_\tau$.
(2) $\mathrm{dom}(\Delta_2)\cup\{x\} \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma_2,\Delta_2)\cup\{x\}$ and for all $\sigma \in [\![\Gamma_2,\Delta_2,\{x:B\}]\!]_\tau$, $t\langle\sigma\rangle \Vdash [\![A]\!]_\tau$.
We immediately obtain that $\mathrm{dom}(\Delta_1,\Delta_2) \subseteq FV(t[\vec{r}/x]) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1,\Gamma_2,\Delta_2)$.

$$t[\vec{r}/x]\langle\sigma\rangle = t[\vec{r}/x]\langle\sigma_1,\sigma_2\rangle = t[\vec{r}\langle\sigma_1\rangle/x]\langle\sigma_2\rangle \Vdash [\![A]\!]_\tau,$$

since $\vec{r}\langle\sigma_1\rangle \Vdash B$ (from 1 and the fact that $\sigma_1 \in [\![\Gamma_1,\Delta_1]\!]_\tau$), and so we have that $\langle\{\vec{r}\langle\sigma_1\rangle/x\},\sigma_2\rangle \in [\![\Gamma_2,\Delta_2,\{x:B\}]\!]_\tau$.

($\forall_i$) Suppose that $\Gamma;\Delta \vdash \vec{t} : A$ is valid and that $X \notin FV(\Gamma,\Delta)$. Then

$$\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma,\Delta) \text{ and } \vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_\tau \text{ for all } \sigma \in [\![\Gamma,\Delta]\!]_\tau$$

Let $\tau' = \tau \setminus \{X\}$. By definition, we have

$$[\![\forall X.A]\!]_{\tau'} = \bigcap_{B\in\mathcal{S}_1}[\![A]\!]_{\tau'\cup\{X\mapsto[\![B]\!]_\emptyset\}}$$

Since $X \notin FV(\Gamma,\Delta)$, for all σ such that $\sigma \in [\![\Gamma,\Delta]\!]_\tau$, we also have $\sigma \in [\![\Gamma,\Delta]\!]_{\tau'\cup\{X\mapsto R\}}$ for any $R \subseteq \mathcal{S}_1$.
Therefore, $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ implies $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_{\tau'\cup\{X\mapsto R\}}$ for all $R \in \mathcal{S}_1$, thus, $\vec{t}\langle\sigma\rangle \Vdash [\![\forall X.A]\!]_{\tau'}$.

($\forall_e$) Suppose that $\Gamma;\Delta \vdash \vec{t} : \forall X.A$. Then

$$\mathrm{dom}(\Delta) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma,\Delta) \text{ and } \vec{t}\langle\sigma\rangle \Vdash [\![\forall X.A]\!]_\tau \text{ for all } \sigma \in [\![\Gamma,\Delta]\!]_\tau$$

Thus, by definition, since $[\![B]\!]_\tau \subseteq \mathcal{S}_1$, we have $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}}$.
We conclude by Lemma 5.2.

□

Lemma 5.5. *For any $\vec{t} \in \mathrm{PUNQ}$ and any $k \geq n$ for $k, n \in \mathbb{N} - \{0\}$,*

$$\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset \iff \begin{cases} \forall i = 0, \ldots, 2^n - 1, \exists \vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset \text{ such that } \vec{t}\,|i\rangle \rightsquigarrow^* \vec{v}_i, \text{ and} \\ \vec{v}_i \perp_{\sharp(\mathbb{B}^k)} \vec{v}_j,\ \forall i \neq j. \end{cases}$$

Proof. (The condition is necessary.) For some $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$, let $\vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$ be the values that satisfy $\vec{t}[|i\rangle/x] \rightsquigarrow^* \vec{v}_i$, $\forall i = 0, \ldots, 2^n - 1$. Then, for any $\alpha_i \in \overline{\mathbb{Q}}$ such that $\sum_i |\alpha_i|^2 = 1$ we have, by linearity,

$$\vec{t}\left[\sum_i \alpha_i \cdot |i\rangle/x\right] = \sum_i \alpha_i \cdot \vec{t}[|i\rangle/x] \rightsquigarrow^* \sum_i \alpha_i \cdot \vec{v}_i.$$

Since $\sum_i \alpha_i \cdot |i\rangle \in [\![\sharp(\mathbb{B}^n)]\!]_\emptyset$, we have that $\sum_i \alpha_i \cdot \vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$, and therefore $\| \sum_i \alpha_i \cdot \vec{v}_i \| = 1$. From this, we can derive

$$\begin{aligned} 1 = \left\| \sum_i \alpha_i \cdot \vec{v}_i \right\|^2 &= \Big\langle \sum_i \alpha_i \cdot \vec{v}_i \Big| \sum_i \alpha_i \cdot \vec{v}_i \Big\rangle \\ &= \sum_i |\alpha_i|^2 \langle \vec{v}_i \mid \vec{v}_i \rangle + \sum_{i<j} \left(\alpha_i \overline{\alpha_j} \langle \vec{v}_i \mid \vec{v}_j \rangle + \alpha_j \overline{\alpha_i} \langle \vec{v}_j \mid \vec{v}_i \rangle \right) \\ &= \sum_i |\alpha_i|^2 + \sum_{i<j} \left(\alpha_i \overline{\alpha_j} \langle \vec{v}_i \mid \vec{v}_j \rangle + \overline{\alpha_i \overline{\alpha_j} \langle \vec{v}_i \mid \vec{v}_j \rangle} \right) \\ &= 1 + \sum_{i<j} 2\mathrm{Re}\left(\alpha_i \overline{\alpha_j} \langle \vec{v}_i | \vec{v}_j \rangle \right). \end{aligned}$$

Picking $\alpha_0 = \alpha_1 = \frac{1}{\sqrt{2}}$, and $\alpha_i = 0$, for $i \neq 0, 1$ we deduce $\mathrm{Re}(\langle \vec{v}_0 \mid \vec{v}_1 \rangle) = 0$ and from $\alpha_0 = \frac{1}{\sqrt{2}}$, $\alpha_1 = \frac{i}{\sqrt{2}}$ and $\alpha_i = 0, \forall i \neq 0, 1$ we obtain $\mathrm{Im}(\langle \vec{v}_0 \mid \vec{v}_1 \rangle) = 0$ and therefore $\langle \vec{v}_0 \mid \vec{v}_1 \rangle = 0$. The same reasoning can be applied to all other pairs α_i, α_j such that $i \neq j$ and therefore $\langle \vec{v}_i \mid \vec{v}_j \rangle = 0$.

(The condition is sufficient.) Suppose there exist $\vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$, $i = 0, \ldots, 2^n - 1$, for which $\langle \vec{v}_i \mid \vec{v}_j \rangle = 0$, $\forall i \neq j$ and such that $\vec{t}[|i\rangle/x] \rightsquigarrow^* \vec{v}_i$. In particular, we have that, for all i, $\vec{v}_i \in \mathrm{span}(\{|i\rangle \mid i = 0, \ldots, 2^n - 1\})$ and $\|\vec{v}_i\| = 1$. For any given $\vec{v} \in [\![\sharp(\mathbb{B}^n)]\!]_\emptyset$, $\vec{v} = \sum_i \alpha_i \cdot |i\rangle$ such that $\sum_i |\alpha_i|^2 = 1$. Then,

$$\vec{t}[\vec{v}/x] = \sum_i \alpha_i \cdot \vec{t}[|i\rangle/x] \rightsquigarrow^* \sum_i \alpha_i \cdot \vec{v}_i \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset,$$

since $\sum_i \alpha_i \cdot \vec{v}_i \in \mathrm{span}(\{|i\rangle \mid i = 0, \ldots, 2^n - 1\})$ and

$$\begin{aligned} \left\| \sum_i \alpha_i \cdot \vec{v}_i \right\|^2 &= \Big\langle \sum_i \alpha_i \cdot \vec{v}_i \Big| \sum_i \alpha_i \cdot \vec{v}_i \Big\rangle \\ &= \sum_i |\alpha_i|^2 \langle \vec{v}_i \mid \vec{v}_i \rangle + \sum_{i<j} \left(\alpha_i \overline{\alpha_j} \langle \vec{v}_i \mid \vec{v}_j \rangle + \alpha_j \overline{\alpha_i} \langle \vec{v}_j \mid \vec{v}_i \rangle \right) \\ &= \sum_i |\alpha_i|^2 \|\vec{v}_i\|^2 + \sum_{i<j} \left(0 + 0 \right) \\ &= \sum_i |\alpha_i|^2 = 1. \end{aligned}$$

Therefore, for all $\vec{v} \in [\![\sharp\mathbb{B}^n]\!]_\emptyset$, $\vec{t}[\vec{v}/x] \Vdash \sharp(\mathbb{B}^n)$, and finally $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)]\!]_\emptyset$. □

Theorem 5.7 (Completeness). *For any isometry $\mathcal{I} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$, with $k \geq n \geq 1$, there exists a closed superposition $\lambda x.\vec{t} \in \mathsf{S}_c$ of type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ that represents $\mathcal{I}$.*

Proof. We provide a constructive proof. If $\mathcal{I} : \overline{\mathbb{Q}}^{2^n} \to \overline{\mathbb{Q}}^{2^k}$ represents an isometry, then the columns in its matrix representation have unit norm and are mutually orthogonal. This means that $\forall i, j = 0, \ldots, 2^n - 1$, $\mathcal{I}(\lfloor |i\rangle \rfloor)^\dagger \cdot \mathcal{I}(\lfloor |j\rangle \rfloor) = 0$ if $i \neq j$. Furthermore, for any $i = 0, \ldots, 2^n - 1$, $\|\mathcal{I}(\lfloor |i\rangle \rfloor)\| = 1$ and so $\mathcal{I}(\lfloor |i\rangle \rfloor) \triangleq (\alpha_0^i \ldots \alpha_{2^k-1}^i)$ can be encoded by a closed PUNQ term $\vec{v}_i = \sum_{j=0}^{2^k-1} \alpha_j^i \cdot |j\rangle \in [\![\sharp(\mathbb{B}^k)]\!]$. We now perform the encoding. Let x, x_m and $x_{[1..m]}$ all represent different variables, for $1 \leq m \leq n$. We can encode the operator $\mathcal{I}$ using nested if statements, as follows:

$$
\begin{aligned}
\vec{t} \triangleq\ & \mathsf{let}\ \left(x_{[1..n-1]}, x_n\right) = x \\
& \quad \mathsf{in\ let}\ \left(x_{[1..n-2]}, x_{n-1}\right) = x_{[1..n-1]} \\
& \qquad \mathsf{in}\ \ldots \\
& \qquad\quad \mathsf{in\ let}\ (x_1, x_2) = x_{[1..2]}\ \mathsf{in\ if}\ x_1\ \ \mathsf{then} \ldots \mathsf{if}\ x_{n-1}\ \mathsf{then}\ (\mathsf{if}\ x_n\ \mathsf{then}\ \vec{v}_0\ \mathsf{else}\ \vec{v}_1) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \ (\mathsf{if}\ x_n\ \mathsf{then}\ \vec{v}_2\ \mathsf{else}\ \vec{v}_3) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else} \ldots \mathsf{if}\ x_{n-1}\ \mathsf{then}\ (\mathsf{if}\ x_n\ \mathsf{then}\ \vec{v}_{2^n-4}\ \mathsf{else}\ \vec{v}_{2^n-3}) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \ (\mathsf{if}\ x_n\ \mathsf{then}\ \vec{v}_{2^n-2}\ \mathsf{else}\ \vec{v}_{2^n-1}).
\end{aligned}
$$

Then, the term $\lambda x.\vec{t} \in \mathsf{S}_c$ has type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$, and $\forall i = 0, \ldots, 2^n - 1$, $(\lambda x.\vec{t})\ |i\rangle \rightsquigarrow^* \vec{v}_i$. □

Theorem 5.8 (Non-separability). *For any $n, k \in \mathbb{N} - \{0\}$, the type $\sharp(\mathbb{B}^{n+k}) \multimap \left(\sharp(\mathbb{B}^n) \times \sharp(\mathbb{B}^k)\right)$ is uninhabited.*

Proof. Consider an argument by contradiction, starting with a candidate closed term $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^{n+k}) \multimap \left(\sharp(\mathbb{B}^n) \times \sharp(\mathbb{B}^k)\right)]\!]_\emptyset$. Define the values $(\vec{u}_i, \vec{v}_i) \in [\![\sharp(\mathbb{B}^n) \times \sharp(\mathbb{B}^k)]\!]_\emptyset$ satisfying $(\lambda x.\vec{t})\ |i\rangle \rightsquigarrow^* (\vec{u}_i, \vec{v}_i)$. Notice that $\lambda x.\vec{t}$ can be subtyped as $\sharp(\mathbb{B}^{n+k}) \multimap \sharp(\mathbb{B}^{n+k})$ and therefore by Theorem 5.6 must satisfy unitarity. We conclude, therefore, that since we have 2^{n+k} values of $(\vec{u}_i, \vec{v}_i)$ such that they are all orthogonal and of unit norm, then they must form an orthonormal basis of $[\![\sharp(\mathbb{B}^{n+k})]\!]_\emptyset$. In particular, they suffice to represent any value $\vec{s} \triangleq \sum_i \alpha_i \cdot (\vec{u}_i, \vec{v}_i)$ with $\sum_i |\alpha_i|^2 = 1$, and in particular we may choose $\vec{s}$ to be an entangled state and therefore $\vec{s} \notin [\![\sharp(\mathbb{B}^n) \times \sharp(\mathbb{B}^k)]\!]_\emptyset$. However, we have that $\sum_i \alpha_i \cdot |i\rangle \in [\![\sharp(\mathbb{B}^{n+k})]\!]_\emptyset$ and that $\lambda x.\vec{t}\ (\sum_i \alpha_i \cdot |i\rangle) \rightsquigarrow^* \vec{s}$. □

A.3 Proofs of Section 6

Shortand notations. For two sets $\mathcal{S}, \mathcal{T}$, we have that

- $\mathcal{S}\,\mathcal{T} \triangleq \{s\ t \mid s \in \mathcal{S},\ t \in \mathcal{T}\}$
- $\lambda z.\mathcal{S} \triangleq \{\lambda z.s \mid s \in \mathcal{S}\}$
- $\mathcal{S}[\mathcal{T}/z] \triangleq \{s[t/z] \mid s \in \mathcal{S},\ t \in \mathcal{T}\}$

Lemma A.2. *Let $\vec{t} \in \mathsf{S}$ and $v \in \mathsf{BV}$, then $(\!|\vec{t}[v/x]|\!) = (\!|\vec{t}\,|\!)[(\!|v|\!)/x]$.*

Proof. By induction on the structure of $\vec{t}$.

- $(\vec{t} = x)$ $(\!|\vec{t}[v/x]|\!) = (\!|v|\!) = x[(\!|v|\!)/x] = (\!|\vec{t}\,|\!)[(\!|v|\!)/x]$.
- $(\vec{t} = y)$ $(\!|\vec{t}[v/x]|\!) = (\!|y|\!) = y = y[(\!|v|\!)/x] = (\!|y|\!)[(\!|v|\!)/x] = (\!|\vec{t}\,|\!)[(\!|v|\!)/x]$.
- $(\vec{t} = |0\rangle)$ $(\!|\,|0\rangle[v/x]|\!) = (\!|\,|0\rangle|\!) = \lambda z_1.\lambda z_2.z_1 = (\lambda z_1.\lambda z_2.z_1)[(\!|v|\!)/x] = (\!|\,|0\rangle|\!)[(\!|v|\!)/x] = (\!|\vec{t}\,|\!)[(\!|v|\!)/x]$.
- $(\vec{t} = |1\rangle)$ Similar to case $(|0\rangle)$.

- $(\vec{t} = \text{if } r \text{ then } \vec{s_1} \text{ else } \vec{s_2})$

$$\begin{aligned}
(\!|\vec{t}[v/x]|\!) &= (\!|\text{if } r[v/x] \text{ then } \vec{s_1}[v/x] \text{ else } \vec{s_2}[v/x]|\!) \\
&= ((\!|r[v/x]|\!)\ \lambda x_{1\ldots k}.(\!|t_1[v/x]|\!)\ \lambda x_{1\ldots k}.(\!|t_2[v/x]|\!))\ x_1 \ldots x_k \\
&=^{\text{IH}} ((\!|r|\!)[(\!|v|\!)/x]\ \lambda x_{1\ldots k}.(\!|t_1|\!)[(\!|v|\!)/x]\ \lambda x_{1\ldots k}.(\!|t_2|\!)[(\!|v|\!)/x])\ x_1 \ldots x_k \\
&= ((\!|r|\!)\ \lambda x_{1\ldots k}.(\!|t_1|\!)\ \lambda x_{1\ldots k}.(\!|t_2|\!))[(\!|v|\!)/x]\ x_1 \ldots x_k \\
&= (((\!|r|\!)\ \lambda x_{1\ldots k}.(\!|t_1|\!)\ \lambda x_{1\ldots k}.(\!|t_2|\!))\ x_1 \ldots x_k)[(\!|v|\!)/x] = (\!|\vec{t}|\!)[(\!|v|\!)/x]
\end{aligned}$$

- $(\vec{t} = \lambda z.\vec{s})$

$$\begin{aligned}
(\!|\vec{t}[v/x]|\!) &= (\!|\lambda z.\vec{s}[v/x]|\!) \\
&= \lambda z.(\!|\vec{s}[v/x]|\!) \\
&=^{\text{IH}} \lambda z.(\!|\vec{s}|\!)[(\!|v|\!)/x] \\
&= (\!|\lambda z.\vec{s}|\!)[(\!|v|\!)/x] = (\!|\vec{t}|\!)[(\!|v|\!)/x].
\end{aligned}$$

- $(\vec{t} = s\ r)$

$$\begin{aligned}
(\!|\vec{t}[v/x]|\!) &= (\!|(s\ r)[v/x]|\!) \\
&= (\!|s[v/x]\ r[v/x]|\!) \\
&= (\!|s[v/x]|\!)\ (\!|r[v/x]|\!) \\
&=^{\text{IH}} (\!|s|\!)[(\!|v|\!)/x]\ (\!|r|\!)[(\!|v|\!)/x] \\
&= ((\!|s|\!)\ (\!|r|\!))[(\!|v|\!)/x] = (\!|s\ r|\!)[(\!|v|\!)/x] = (\!|\vec{t}|\!)[(\!|v|\!)/x].
\end{aligned}$$

- $(\vec{t} = (s, r))$

$$\begin{aligned}
(\!|\vec{t}[v/x]|\!) &= (\!|(s, r)\ [v/x]|\!) \\
&= (\!|(s[v/x], r[v/x])|\!) \\
&= \lambda z.(z\ (\!|s[v/x]|\!)\ (\!|r[v/x]|\!)) \\
&=^{\text{IH}} \lambda z.(z\ (\!|s|\!)[(\!|v|\!)/x]\ (\!|r|\!)[(\!|v|\!)/x]) \\
&= (\lambda z.(z\ (\!|s|\!)\ (\!|r|\!)))[(\!|v|\!)/x] = (\!|(s, r)|\!)[(\!|v|\!)/x] = (\!|\vec{t}|\!)[(\!|v|\!)/x].
\end{aligned}$$

- $(\vec{t} = \text{let } (z_1, z_2) = s \text{ in } \vec{r})$

$$\begin{aligned}
(\!|\vec{t}[v/x]|\!) &= (\!|\text{let } (z_1, z_2) = s[v/x] \text{ in } \vec{r}[v/x]|\!) \\
&= (\!|s[v/x]|\!)(\lambda z_1.\lambda z_2.(\!|\vec{r}[v/x]|\!)) \\
&=^{\text{IH}} (\!|s|\!)[(\!|v|\!)/x](\lambda z_1.\lambda z_2.(\!|\vec{r}|\!)[(\!|v|\!)/x]) \\
&= \big((\!|s|\!)(\lambda z_1.\lambda z_2.(\!|\vec{r}|\!))\big)[(\!|v|\!)/x] \\
&= (\!|\text{let } (z_1, z_2) = s \text{ in } \vec{r}|\!)[(\!|v|\!)/x] = (\!|\vec{t}|\!)[(\!|v|\!)/x].
\end{aligned}$$

- $(\vec{t} = \vec{0})$ Similar to case of $|0\rangle$.
- $(\vec{t} = \alpha \cdot \vec{s})$ $(\!|\vec{t}[v/x]|\!) = (\!|\alpha \cdot \vec{s}[v/x]|\!) = \text{if } \alpha = 0$
- $(\vec{t} = \vec{s} + \vec{r})$ □

Lemma 6.6. *If $\vec{t} \equiv \vec{r}$, then $(\!|\vec{t}|\!) \simeq (\!|\vec{r}|\!)$.*

Proof. By inspection of the equivalence relation.

- $\vec{t_1} + \vec{t_2} \equiv \vec{t_2} + \vec{t_1}$, where we have $(\!|\vec{t_1} + \vec{t_2}|\!) = (\!|\vec{t_1}|\!) \cup (\!|\vec{t_2}|\!) = (\!|\vec{t_2}|\!) \cup (\!|\vec{t_1}|\!) = (\!|\vec{t_2} + \vec{t_1}|\!)$.

- $\vec{0} + \vec{t}_1 \equiv \vec{t}_1$, in which case $(\!|\vec{0} + \vec{t}_1|\!) = \{*\} \cup (\!|\vec{t}_1|\!)$, therefore $(\!|\vec{0} + \vec{t}_1|\!) \simeq (\!|\vec{t}_1|\!)$.

For all other cases, we easily find that $(\!|\vec{t}|\!) = (\!|\vec{r}|\!)$ and the conclusion follows. □

Lemma 6.7. $\Gamma; \Delta \vdash \vec{t} : A$ *implies* $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!) : A^\star$.

Proof. We prove this statement by induction on the derivation tree of $\Gamma; \Delta \vdash \vec{t} : A$.

(W) By the induction hypothesis, we have that $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!) : A^\star$. Using rule (Weak) in DLAL we obtain $\Gamma^\star, \Gamma'^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!) : A^\star$.

(C) By the IH we have $\Gamma^\star, x : B^\star, y : B^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!) : A^\star$. Applying rule (Cntr) in DLAL we obtain $\Gamma^\star, x : B^\star; \Delta^\star \vdash_{\mathrm{DLAL}} (\!|\vec{t}|\!)[x/y] : A^\star$ which by Lemma A.2

(≡) We can check that this is the case for all terms $\vec{t}_1, \vec{t}_2$ such that $\vec{t}_1 \equiv \vec{t}_2$. It can be shown by inspection that, for almost all cases in the equivalence relation ≡, the translation is preserved, i.e. that if $\vec{t}_1 \equiv \vec{t}_2$, then $(\!|\vec{t}_1|\!) = (\!|\vec{t}_2|\!)$. However, this is not the case in two instances:
- $\vec{t}_1 = \vec{0} + \vec{t} \equiv \vec{t} = \vec{t}_2$. In this case, $(\!|\vec{t}_1|\!) = (\!|\vec{t}_2|\!) \cup \{*\}$ and therefore $(\!|\vec{t}_2|\!) \subseteq (\!|\vec{t}_1|\!)$ from which the conclusion follows.
- $\vec{t}_1 = 0 \cdot \vec{t} \equiv \vec{0} = \vec{t}_2$. Here, since $(\!|\vec{t}_2|\!) = \{*\}$ which admits any type, the conclusion follows.

(≤) By Lemma 6.4, for all subtying relations $A \leq B$ we have that $A^\star = B^\star$ and therefore the result follows.

(Ax) We have that $(\!|x|\!) = \{x\}$, and by rule (Id) in DLAL, we may obtain $; x : A^\star \vdash x : A^\star$ for any $A \in \mathbb{T}$.

(0) We have that $(\!|\,|0\rangle\,|\!) = \{\lambda x.\lambda y.x\}$ and $\mathbb{B}^\star = \forall X.(X \multimap X \multimap X)$, and by DLAL rules (Id), (Weak), and two uses of (⊸ i), and finally (∀ i) we obtain the desired typing derivation. Indeed, this is the standard encoding in DLAL for booleans.

(1) Analogous to (0).

(if) We have that $(\!|\,\text{if } t \text{ then } t_1 \text{ else } t_2\,|\!) = (t\ \lambda x_{1\ldots k}.(\!|t_1|\!)\ \lambda x_{1\ldots k}.(\!|t_2|\!))\ x'_1 \ldots x'_k$. By the IH we have that $\Gamma'^\star; \Delta'^\star \vdash (\!|t_1|\!) \cup (\!|t_2|\!) : A^\star$, and $\Gamma^\star; \Delta^\star \vdash t : \forall X.(X \multimap X \multimap X)$. Let $\Delta'^\star = \{x_i : A_i^\star\}_{i=1\ldots k}$,

$$\frac{\Gamma'^\star; \Delta'^\star \vdash (\!|t_1|\!) \cup (\!|t_2|\!) : A^\star}{\Gamma'^\star; \vdash (\!|\lambda x_{1\ldots k}.t_1|\!) \cup (\!|\lambda x_{1\ldots k}.t_2|\!) : A_1^\star \multimap \cdots \multimap A_k^\star \multimap A^\star}\ (\multimap \text{e})\ k \text{ times}$$

Without loss of generality, we may consider that the variables in $\Gamma'^\star$ have different names in the typing of t_1 and t_2. Let us use $\widetilde{\Gamma'^\star}$ to denote the nonlinear context of $\widetilde{t_2}$, where we have renamed the nonlinear variables.
Therefore, we may derive the following judgement in DLAL. Let $B \triangleq A_1^\star \multimap \cdots \multimap A_k^\star \multimap A^\star$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma^\star; \Delta^\star \vdash t : \forall X.(X \multimap X \multimap X)}{\Gamma^\star; \Delta^\star \vdash t : B \multimap B \multimap B}\ (\forall \text{ e}) \qquad \Gamma'^\star; \vdash (\!|\lambda x_{1\ldots k}.t_1|\!) : B}{\Gamma^\star, \Gamma'^\star; \Delta^\star \vdash t\ (\!|\lambda x_{1\ldots k}.t_1|\!) : B \multimap B}\ (\multimap \text{e}) \qquad \widetilde{\Gamma'^\star}; \vdash (\!|\lambda x_{1\ldots k}.\widetilde{t_2}|\!) : B}{\Gamma^\star, \Gamma'^\star, \widetilde{\Gamma'^\star}; \Delta^\star \vdash t\ (\!|\lambda x_{1\ldots k}.t_1|\!)\ (\!|\lambda x_{1\ldots k}.\widetilde{t_2}|\!) : B}\ (\multimap \text{e})}{\Gamma^\star, \Gamma'^\star; \Delta^\star \vdash t\ (\!|\lambda x_{1\ldots k}.t_1|\!)\ (\!|\lambda x_{1\ldots k}.t_2|\!) : A_1^\star \multimap \cdots \multimap A_k^\star \multimap A^\star}\ (\text{Cntr}) \qquad \dfrac{}{x_i : A_i^\star \vdash x_i : A_i^\star}\ (\text{Id})}{\Gamma^\star, \Gamma'^\star; \Delta^\star, \Delta'^\star \vdash (t\ (\!|\lambda x_{1\ldots k}.t_1|\!)\ (\!|\lambda x_{1\ldots k}.t_2|\!))\ x_1 \ldots x_k : A^\star}\ (\multimap \text{e})$$

(if$_\sharp$) Since $(\sharp A)^\star = A^\star$, the proof is precisely the same as done in the case of (if).

($\multimap_i$) By the IH we have that $\Gamma^\star; \Delta^\star, x : A^\star \vdash (\!|\vec{t}|\!) : B^\star$. By rule (⊸ i) in DLAL we obtain $\Gamma^\star; \Delta^\star \vdash \lambda x.(\!|\vec{t}|\!) : A^\star \multimap B^\star$. Since $(\!|\lambda x.\vec{t}|\!) = \lambda x.(\!|\vec{t}|\!)$ and $(A \multimap B)^\star = A^\star \multimap B^\star$ this is precisely our desired conclusion.

($\multimap_e$) By the IH we have $\Gamma^\star;\Delta^\star \vdash (\!|t|\!) : A^\star \multimap B^\star$ and $\Gamma'^\star;\Delta'^\star \vdash (\!|s|\!) : A^\star$. Applying the DLAL rule ($\multimap$ e) we obtain $\Gamma^\star,\Gamma'^\star;\Delta^\star,\Delta'^\star \vdash (\!|t|\!)\,(\!|s|\!) : B^\star$. Since $(\!|t\ s|\!) = (\!|t|\!)\,(\!|s|\!)$ the proof is concluded.

($\Rightarrow_i$) By the IH we have $\Gamma^\star, x : A^\star;\Delta^\star \vdash (\!|\vec{t}|\!) : B^\star$. By rule ($\Rightarrow$ i) we obtain $\Gamma^\star;\Delta^\star \vdash \lambda x.(\!|\vec{t}|\!) : A^\star \Rightarrow B^\star$. By Lemma 6.3 we have that $(!(A))^\star = A^\star$.

($\Rightarrow_e$) Similar to ($\multimap_e$). We may simply use the rule ($\Rightarrow$ e) in DLAL to obtain our desired conclusion.

($\times_i$) By the IH we have that $\Gamma^\star;\Delta^\star \vdash (\!|t|\!) : A^\star$ and $\Gamma'^\star;\Delta'^\star \vdash (\!|s|\!) : B^\star$. We may obtain the following derivation in DLAL (i.e. the pair encoding).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{;x : A^\star \multimap B^\star \multimap X \vdash x : A^\star \multimap B^\star \multimap X}\,(\text{Id}) \qquad \Gamma^\star;\Delta^\star \vdash (\!|t|\!) : A^\star}{\Gamma^\star;\Delta^\star, x : A^\star \multimap B^\star \multimap X \vdash x\,(\!|t|\!) : B^\star \multimap X}\,(\multimap \text{e}) \qquad \Gamma'^\star;\Delta'^\star \vdash (\!|s|\!) : B^\star}{\Gamma^\star,\Gamma'^\star;\Delta^\star,\Delta'^\star, x : A^\star \multimap B^\star \multimap X \vdash x\,(\!|t|\!)\,(\!|s|\!) : X}\,(\multimap \text{e})}{\Gamma^\star,\Gamma'^\star;\Delta^\star,\Delta'^\star \vdash \lambda x.(x\,(\!|t|\!)\,(\!|s|\!)) : (A^\star \multimap B^\star \multimap X) \multimap X}\,(\multimap \text{i})}{\Gamma^\star,\Gamma'^\star;\Delta^\star,\Delta'^\star \vdash \lambda x.(x\,(\!|t|\!)\,(\!|s|\!)) : \forall X.((A^\star \multimap B^\star \multimap X) \multimap X)}\,(\forall \text{ i})$$

This is precisely our desired result.

($\times_e$) We have $\Gamma^\star;\Delta^\star \vdash (\!|t|\!) : \forall X.((A^\star \multimap B^\star \multimap X) \multimap X)$ and $\Gamma'^\star;\Delta'^\star, x : A^\star, y : B^\star \vdash (\!|\vec{s}|\!) : C^\star$ by the IH. In DLAL we are able to derive

$$\dfrac{\dfrac{\Gamma^\star;\Delta^\star \vdash (\!|t|\!) : \forall X.((A^\star \multimap B^\star \multimap X) \multimap X)}{\Gamma^\star;\Delta^\star \vdash (\!|t|\!) : (A^\star \multimap B^\star \multimap C^\star) \multimap C^\star}\,(\forall \text{ e}) \qquad \dfrac{\dfrac{\dfrac{\Gamma'^\star;\Delta'^\star, x : A^\star, y : B^\star \vdash (\!|\vec{s}|\!) : C^\star}{\Gamma'^\star;\Delta'^\star, x : A^\star \vdash \lambda y.(\!|\vec{s}|\!) : B^\star \multimap C^\star}\,(\multimap \text{i})}{\Gamma'^\star;\Delta'^\star \vdash \lambda x.\lambda y.(\!|\vec{s}|\!) : A^\star \multimap B^\star \multimap C^\star}\,(\multimap \text{i})}{\Gamma^\star,\Gamma'^\star;\Delta^\star,\Delta'^\star \vdash (\!|t|\!)\ \lambda x.\lambda y.(\!|\vec{s}|\!) : C^\star}\,(\multimap \text{e})$$

which concludes this case.

($\times_{e\sharp}$) Same proof as in ($\times_e$).

($\sharp_i$) Trivial.

($\S_i$) ($\S_e$) ($\forall_i$) ($\forall_e$) These rules have a direct counterpart in DLAL so their proof is trivial.

□

Lemma 6.8. *For any $A \in \mathbb{T}$, there exists a polynomial P_A such that, for any $\vec{t}$ where $;\vdash \vec{t} : A$, the following holds $|(\!|\vec{t}|\!)| = O(P_A(|\vec{t}|))$.*

Proof. By induction on the structure of $\vec{t}$.

- If $\vec{t} = |0\rangle$, then $(\!|\vec{t}|\!) = \{\lambda x.\lambda y.x\}$ and therefore $|\vec{t}| = 1$ and $|(\!|\vec{t}|\!)| = 5$. Similar case for $t = |1\rangle$.
- If $\vec{t} = x$, then $|t| = 1$ and $(\!|\vec{t}|\!) = \{x\}$ and therefore $|(\!|\vec{t}|\!)| = 1$.
- If $\vec{t} = \lambda x.\vec{s}$, then $|\vec{t}| = 1 + |\vec{s}|$ and $|(\!|\vec{t}|\!)| = |\lambda x_{[1\ldots k]}.(\!|\vec{s}_{[x:1\ldots k]}|\!)| = O(2|\vec{s}|) = O(|\vec{t}|)$.
- If $\vec{t} = t_1\ t_2$, we have $|\vec{t}| = |t_1| + |t_2|$ and

$$\begin{aligned}|(\!|t_1\ t_2|\!)| &\triangleq |(\ldots((\!|t_1|\!)\,(\!|t_2|\!))\ \ldots\,(\!|t_2|\!))| \\ &= |(\!|t_1|\!)| + O(|t_1|)\cdot|(\!|t_2|\!)| \\ &=^{\text{IH}} O(|t_1|^{d_1}) + O(|t_1|)\cdot O(|t_2|^{d_2}) \\ &= O(|\vec{t}|^{\max(d_1,d_2+1)}).\end{aligned}$$

Notice that, if t_1 does not duplicate t_2, then the bounding degree is instead $\max(d_1, d_2)$.

- If $\vec{t} = \text{if } s \text{ then } p_1 \text{ else } p_2$, we have $|\vec{t}| = 1 + |s| + |p_1| + |p_2|$, and

$$\begin{aligned}|(\!|\vec{t}|\!)| \triangleq& \left|((\!|s|\!)\ \lambda x_{[1\ldots k]}.(\!|p_1|\!)\ \lambda x_{[1\ldots k]}.(\!|p_2|\!))\ x'_1 \ldots x'_k\right| \\ =&|(\!|s|\!)| + O(|p_1|) + O(|(\!|p_1|\!)|) + O(|p_2|) + O(|(\!|p_2|\!)|) \\ =^{\text{IH}}& O(|s|^{d_1}) + O(|p_1|^{d_2}) + O(|p_2|^{d_3}) + O(|p_1|) + O(|p_2|) \\ =&O(|\vec{t}|^{\max(d_1,d_2,d_3)}).\end{aligned}$$

- If $\vec{t} = (t_1, t_2)$, then $|\vec{t}| = 1 + |t_1| + |t_2|$ and

$$\begin{aligned}|(\!|\vec{t}|\!)| \triangleq |\lambda x.(x\ (\!|t_1|\!)\ (\!|t_2|\!))| = 3 + |(\!|t_1|\!)| + |(\!|t_2|\!)| =^{\text{IH}}& 2 + O(|t_1|^{d_1}) + O(|t_2|^{d_2}) \\ =& O(|\vec{t}|^{\max(d_1,d_2)}).\end{aligned}$$

- If $\vec{t} = \text{let } (x, y) = t_1 \text{ in } t_2$ then $|t| = 1 + \max(|t_1|, |t_2|)$ and

$$\begin{aligned}|(\!|\vec{t}|\!)| \triangleq |(\!|t_1|\!)\ (\lambda x.\lambda y.(\!|t_2|\!))| &= 2 + |(\!|t_1|\!)| + |(\!|t_2|\!)| \\ &=^{\text{IH}} 2 + O(|t_1|^{d_1}) + O(|t_2|^{d_2}) \\ &= O(|\vec{t}|^{\max(d_1,d_2)})\end{aligned}$$

- If $\vec{t} = \vec{t}_1 + \vec{t}_2$, then $|\vec{t}| = \max(|\vec{t}_1|, |\vec{t}_2|)$, and

$$\begin{aligned}|(\!|\vec{t}|\!)| = |(\!|\vec{t}_1|\!) \cup (\!|\vec{t}_2|\!)| = \max(|(\!|\vec{t}_1|\!)|, |(\!|\vec{t}_2|\!)|) =^{\text{IH}}& O(\max(|\vec{t}_1|^{d_1}, |\vec{t}_2|^{d_2})) \\ =& O(|\vec{t}|^{\max(d_1,d_2)}).\end{aligned}$$

- If $\vec{t} = \vec{t}_1 + \vec{t}_2$, then $|\vec{t}| = \max(|\vec{t}_1|, |\vec{t}_2|)$, and

$$\begin{aligned}|(\!|\vec{t}|\!)| = |(\!|\vec{t}_1|\!) \cup (\!|\vec{t}_2|\!)| = \max(|(\!|\vec{t}_1|\!)|, |(\!|\vec{t}_2|\!)|) =^{\text{IH}}& O(\max(|\vec{t}_1|^{d_1}, O(|\vec{t}_2|^{d_2})) \\ =& O(|\vec{t}|^{\max(d_1,d_2)}).\end{aligned}$$

- If $\vec{t} = \alpha \cdot \vec{s}$, then $|\vec{t}| = |\vec{s}|$, and $|(\!|\vec{t}|\!)| = |(\!|\vec{s}|\!)| =^{\text{IH}} O(|\vec{s}|^d) = O(|\vec{t}|^d)$.

The degree of the polynomial bounding the size of $(\!|\vec{t}|\!)$ is only increased in translating duplications (see case $\vec{t} = t_1\ t_2$), so it will only depend on the number of such duplications present in the term, which is bounded by the type of the term. □

Lemma 6.9. *For any $\vec{v} \in \mathsf{V}_c$, we have that $(\!|\vec{v}|\!)$ does not reduce.*

Proof. By induction on the structure of $\vec{v}$.

- $\vec{v} = |0\rangle$. Then $(\!|\vec{v}|\!) = \{\lambda x.\lambda y.x\}$, which does not reduce. Same for $\vec{v} = |1\rangle$.
- $\vec{v} = \lambda x.\vec{t}$. Then $(\!|\vec{v}|\!) = \lambda x.(\!|\vec{t}|\!)$ which will not reduce in DLAL in call by value.
- $\vec{v} = (v_1, v_2)$, in which case $(\!|\vec{v}|\!) = \lambda x.(x\ (\!|v_1|\!)\ (\!|v_2|\!))$. By IH, we consider that no element in $(\!|v_1|\!) \cup (\!|v_2|\!)$ reduces in DLAL, therefore $(\!|\vec{v}|\!)$ will not reduce.
- $\vec{v} = \vec{0}$, where we have $(\!|\vec{v}|\!) = \{*\}$, which by definition does not reduce.
- $\vec{v} = \alpha \cdot \vec{w}$, where we apply the IH to $(\!|\vec{w}|\!) = (\!|\vec{v}|\!)$.
- $\vec{v} = \vec{v}_1 + \vec{v}_2$, where we apply the IH to $(\!|\vec{v}_1|\!)$ and $(\!|\vec{v}_2|\!)$, since $(\!|\vec{v}|\!) = (\!|\vec{v}_1|\!) \cup (\!|\vec{v}_2|\!)$. □

Lemma 6.10. *For any $\vec{t} \in \text{PUNQ}$, $\vec{t} \rightsquigarrow \vec{r}$ implies $(\!|\vec{r}|\!) \sqsubseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|\vec{t}|\!) \to^n \mathcal{S}_n\}$.*

Proof. By induction on the reduction of $\vec{t}$.

(If$_0$) $\vec{t} = \text{if } |0\rangle \text{ then } \vec{s_0} \text{ else } \vec{s_1} \rightsquigarrow \vec{s_0}$, therefore $(\!|\vec{r}|\!) = (\!|\vec{s_0}|\!)$ and we have

$$\begin{aligned}(\!|\vec{t}|\!) =& ((\lambda x.\lambda y.x)\ \lambda x_{1\ldots k}.(\!|\vec{s_0}|\!)\ \lambda x_{1\ldots k}.(\!|\vec{s_1}|\!))\ x_1 \ldots x_k\\ \to& ((\lambda y.\lambda x_{1\ldots k}.(\!|\vec{s_0}|\!))\ \lambda x_{1\ldots k}.(\!|\vec{s_1}|\!))\ x_1 \ldots x_k\\ \to& (\lambda x_{1\ldots k}.(\!|\vec{s_0}|\!))\ x_1 \ldots x_k\\ \to^k& (\!|\vec{s_0}|\!)\end{aligned}$$

(If$_1$) $\vec{t} = \text{if } |1\rangle \text{ then } \vec{s_0} \text{ else } \vec{s_1} \rightsquigarrow \vec{s_1}$, therefore $(\!|\vec{r}|\!) = (\!|\vec{s_1}|\!)$ and we have

$$\begin{aligned}(\!|\vec{t}|\!) =& ((\lambda x.\lambda y.x)\ \lambda x_{1\ldots k}.(\!|\vec{s_0}|\!)\ \lambda x_{1\ldots k}.(\!|\vec{s_1}|\!))\ x_1 \ldots x_k\\ \to& ((\lambda y.y)\ \lambda x_{1\ldots k}.(\!|\vec{s_1}|\!))\ x_1 \ldots x_k\\ \to& (\lambda x_{1\ldots k}.(\!|\vec{s_1}|\!))\ x_1 \ldots x_k\\ \to^k& (\!|\vec{s_1}|\!)\end{aligned}$$

(Abs) $\vec{t} = (\lambda x.\vec{p})\ v \rightsquigarrow \vec{p}[v/x]$, and therefore

$$\begin{aligned}(\!|\vec{t}|\!) = (\!|(\lambda x.\vec{p})\ v|\!) &= \lambda x_1 \ldots \lambda x_k.(\!|\vec{p}_{[x:1\ldots k]}|\!)\ \overbrace{(\!|v|\!)\ \ldots\ (\!|v|\!)}^{k \text{ times}}\\ &= \{(\lambda x_1 \ldots \lambda x_k.p_i)\ v_1\ \ldots v_k \mid p_i \in (\!|\vec{p}_{[x:1\ldots k]}|\!),\ v_j \in (\!|v|\!), j = 1 \ldots k\}\\ &\to \{p_i[v_1/x_1, \ldots, v_k/x_k] \mid p_i \in (\!|\vec{p}|\!),\ v_j \in (\!|v|\!), j = 1 \ldots k\}\\ &= (\!|\vec{p}|\!)[(\!|v|\!)/x] = (\!|\vec{p}[v/x]|\!) = (\!|\vec{r}|\!)\end{aligned}$$

(Let) $\vec{t} = (\text{let } (x,y) = (v,w) \text{ in } \vec{s})$, then $\vec{t} \rightsquigarrow \vec{s}[v/x, w/y] = \vec{s}[v/x][w/y]$, since $y \notin FV(v)$.

$$\begin{aligned}(\!|\vec{t}|\!) &= \lambda z.(z\ (\!|v|\!)\ (\!|w|\!))\ \lambda x.\lambda y.(\!|\vec{s}|\!)\\ &\to (\lambda x.\lambda y.(\!|\vec{t}|\!))\ (\!|v|\!)\ (\!|w|\!)\\ &\to (\lambda y.(\!|\vec{s}|\!)[(\!|v|\!)/x])(\!|w|\!)\\ &\to (\!|\vec{s}|\!)[(\!|v|\!)/x][(\!|w|\!)/y]\\ &= (\!|\vec{s}[v/x]|\!)[(\!|w|\!)/y] && \text{(by Lemma A.2)}\\ &= (\!|\vec{s}[v/x][w/y]|\!) && \text{(by Lemma A.2)}\\ &= (\!|\vec{r}|\!), && \text{since } y \notin FV(v).\end{aligned}$$

(If$_+$) If $s_1 \rightsquigarrow \vec{s}_2$, then $\vec{t} = (\text{if } s_1 \text{ then } \vec{p}_1 \text{ else } \vec{p}_2) \rightsquigarrow (\text{if } \vec{s}_2 \text{ then } \vec{p}_1 \text{ else } \vec{p}_2) = \vec{r}$. By the induction hypothesis, $(\!|\vec{s}_2|\!) \subseteq \bigcup_{n>0}\{\mathcal{R}_n \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}$.

$$\begin{aligned}(\!|\vec{r}|\!) = (\!|\text{if } \vec{s_2} \text{ then } \vec{p}_1 \text{ else } \vec{p}_2|\!) &= (\!|\vec{s_2}|\!)\ (\!|\vec{p}_1|\!)\ (\!|\vec{p}_2|\!)\\ &\sqsubseteq^{\text{IH}} \bigcup_{n>0}\{\mathcal{R}_n\ (\!|\vec{p}_1|\!)\ (\!|\vec{p}_2|\!) \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}\\ &\subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|s_1|\!)\ (\!|\vec{p}_1|\!)\ (\!|\vec{p}_2|\!) \to^n \mathcal{S}_n\}\\ &= \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|s_1\ \vec{p}_1\ \vec{p}_2|\!) \to^n \mathcal{S}_n\}\end{aligned}$$

(App) Let $(\!|\vec{s}_2|\!) \subseteq \bigcup_{n>0}\{\mathcal{R}_n \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}$, then

$$\begin{aligned}(\!|\vec{r}|\!) = (\!|p\ \vec{s}_2|\!) &= (\!|p|\!)\ (\!|\vec{s}_2|\!)\\ &\sqsubseteq^{\mathrm{IH}} \bigcup_{n>0}\{(\!|p|\!)\ \mathcal{R}_n \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}\\ &\subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|p|\!)\ (\!|s_1|\!) \to^n \mathcal{S}_n\}\\ &= \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|p\ s_1|\!) \to^n \mathcal{S}_n\}\end{aligned}$$

($\mathrm{App}_\vee$) Similar to (App).

(Pair) Let $(\!|\vec{s}_2|\!) \subseteq \bigcup_{n>0}\{\mathcal{R}_n \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}$, then

$$\begin{aligned}(\!|\vec{r}|\!) = (\!|(\vec{s}_2, p)|\!) &= \lambda x.(x\ (\!|\vec{s}_2|\!)\ (\!|p|\!))\\ &\sqsubseteq^{\mathrm{IH}} \bigcup_{n>0}\{\lambda x.(x\ \mathcal{R}_n\ (\!|p|\!)) \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}\\ &\subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid \lambda x.(x\ (\!|s_1|\!)\ (\!|p|\!)) \to^n \mathcal{S}_n\}\\ &= \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|(s_1, p)|\!) \to^n \mathcal{S}_n\}\end{aligned}$$

($\mathrm{Pair}_\vee$) Similar to (Pair).

(Let_+) Let $\vec{t} = (\mathsf{let}\ \ (x, y) = s_1\ \mathsf{in}\ \vec{p})$ such that $s_1 \rightsquigarrow \vec{s}_2$. Then

$$\begin{aligned}(\!|\vec{r}|\!) &= (\!|\vec{s}_2|\!)\ \lambda x.\lambda y.(\!|\vec{p}|\!)\\ &\sqsubseteq^{\mathrm{IH}} \bigcup_{n>0}\{\mathcal{R}_n\ \lambda x.\lambda y.(\!|\vec{p}|\!) \mid (\!|s_1|\!) \to^n \mathcal{R}_n\}\\ &\subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|s_1|\!)\ \lambda x.\lambda y.(\!|\vec{p}|\!) \to^n \mathcal{S}_n\}\\ &= \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|s_1\ \lambda x.\lambda y.\vec{p}|\!) \to^n \mathcal{S}_n\}\end{aligned}$$

(Sup) Let $\vec{t} = \sum_{i\in I}\alpha_i \cdot p_i + \sum_{j\in J}\beta_j \cdot v_j$ such that $\forall i \in I,\ p_i \rightsquigarrow s_i$. Then, by the induction hypothesis, we have that $\forall i \in I,\ (\!|s_i|\!) \subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|p_i|\!) \to^n \mathcal{S}_n\}$. We can see that

$$\begin{aligned}(\!|\vec{r}|\!) &= \bigcup_{i\in I}(\!|s_i|\!) \cup \bigcup_{j\in J}(\!|v_j|\!)\\ &\sqsubseteq^{\mathrm{IH}} \bigcup_{i\in I}\left(\bigcup_{n>0}\{\mathcal{S}_n \mid (\!|p_i|\!) \to^n \mathcal{S}_n\}\right) \cup \bigcup_{j\in J}(\!|v_j|\!)\\ &= \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|\vec{t}|\!) \to^n \mathcal{S}_n\}. \qquad \text{(by Lemma 6.9)}\end{aligned}$$

(Equ) Since we have that $\vec{s_1} \equiv \vec{s_2}$ and $\vec{s_2} \rightsquigarrow \vec{s_3}$ and $\vec{s_3} \equiv \vec{s_4}$, by the induction hypothesis the following are true:

$$\begin{aligned}&\text{(IH1)} \quad (\!|\vec{s_1}|\!) \simeq (\!|\vec{s_2}|\!),\\ &\text{(IH2)} \quad (\!|\vec{s_3}|\!) \sqsubseteq \bigcup_{n>0}\{\mathcal{S}_n \mid (\!|\vec{s_2}|\!) \to^n \mathcal{S}_n\},\\ &\text{(IH3)} \quad (\!|\vec{s_3}|\!) \simeq (\!|\vec{s_4}|\!).\end{aligned}$$

We may then see that:

$$(\!|\vec{s_4}|\!) \simeq^{\text{IH3}} (\!|\vec{s_3}|\!) \sqsubseteq^{\text{IH2}} \bigcup_{n>0} \{\mathcal{S}_n \mid (\!|\vec{s_2}|\!) \to^n \mathcal{S}_n\} \simeq^{\text{IH1}} \bigcup_{n>0} \{\mathcal{S}_n \mid (\!|\vec{s_1}|\!) \to^n \mathcal{S}_n\}.$$

This concludes the proof. □

A.4 Proofs of Section 7

In this section of the Appendix we consider the proofs and general points addressed in Section 7.

Restriction to the algebraic numbers. It is important to note the role that the complex coefficients in PUNQ play in the decidability of the typing system. In PUNQ, the restriction of amplitudes to the set $\overline{\mathbb{Q}}$ of algebraic numbers ensures that, for any amplitude α, we are capable of checking if $\alpha = 0$ and therefore, for some $\vec{t}$, if $\alpha \cdot \vec{t} \equiv \vec{0}$. This can be done efficiently when $\alpha \in \overline{\mathbb{Q}}$, as shown in [Halava et al. 2005, Proposition 2.2]. Allowing for the full set of complex numbers, on the other hand, allows for the encoding of undecidable problems, as shown in [Adleman et al. 1997].

Reducing ground types. In this section, we will be interested in the time complexity of reducing terms in order to check if they are orthogonal (see Section 3.2). In the program semantics (Figure 4), we allow in rule (Sup) for one-step reductions to alter different parts of a superposition simultaneously, so as to capture a single-step evolution of a quantum machine. However, if type checking requires performing some reduction steps, it must do so in the *classical* model, and therefore, we must consider the number of steps required by a classical Turing machine to perform the reduction.

From the typing rules (Figure 6), we are only interested in checking the orthogonality of terms to which we have successfully attributed some ground type Q. This limitation ensures a bounded dimension in the set of realizers, and a maximum number of base values that can be in a superposition. For such a Q, we define this number the *cardinality of* Q.

Definition A.3 (Ground type dimension). For any ground type Q, we define the *cardinality of* Q, written $\text{card}(Q)$, inductively as:

$$\text{card}(\mathbb{B}) \triangleq 2, \quad \text{card}(\sharp Q) \triangleq \text{card}(Q), \quad \text{card}(\S Q) \triangleq \text{card}(Q), \quad \text{card}(Q \times R) \triangleq \text{card}(Q)\text{card}(R).$$

Using the unitary semantics of Section 5, we can, in an equivalent way, define $\text{card}(Q)$ as the cardinality of $\flat[\![Q]\!]_\emptyset$.

Lemma A.4. *Let* $;\vdash \vec{t} : Q$ *such that* $\vec{t} \rightsquigarrow^* \vec{v}$ *for* $\vec{v} \in \mathsf{V}$. *Then* $\vec{v}$ *can be computed from* $\vec{t}$ *classically in time at most* $O(\text{card}(Q)\,\text{poly}(|\vec{t}|))$.

Proof. Since $\vec{t} \in [\![Q]\!]_\emptyset$, by the unitarity semantics, we have that $\vec{t} \equiv \sum_{i=1}^n \alpha_i \cdot t_i$, such that $\forall i, t_i \in \flat[\![Q]\!]_\emptyset$ and $n \leq \dim(Q)$. The treatment of each t_i in a reduction step is at most quadratic on $|t_i|$ (e.g. checking if t_i is a value, performing variable substitution, converting to normal form, …). This is done for each term t_i. Since there are at most n such t_i, where n is a constant fixed by the type Q, the number of reduction steps in the classical model remains polynomial. □

Lemma 7.5. $\mathsf{CHK}_\emptyset \subseteq \mathsf{CHK}_\times \subseteq \mathsf{CHK}_{\text{untyped}} \subseteq \mathsf{CHK}$.

Proof. The fact that $\mathsf{CHK}_\emptyset \subseteq \mathsf{CHK}_\times$ is trivial. Similarly, $\mathsf{ORTHO}_\forall \subseteq \mathsf{ORTHO}$ since the only difference is that we perform an orthogonality check over all values and not the subset of values that correspond to the correct type. We also have that $\mathsf{ORTHO}_\times \subseteq \mathsf{ORTHO}_\forall$, since any variable substitution will still lead to orthogonal terms since the closed subterm which ensures orthogonality will not be altered. □

Lemma 7.6. $\mathsf{ORTHO}_\emptyset, \mathsf{ORTHO}_\times \in \mathsf{PTIME}$, *and* $\mathsf{ORTHO}_{\mathsf{untyped}} \in \Pi_1^0$.

Proof. We analyze each case separately.

- ($\mathsf{ORTHO}_\emptyset \in \mathsf{PTIME}$) Since $; \vdash \vec{t}, \vec{s} : Q$, by Theorem 6.12, there exists a polynomial P_Q such that $\vec{t}$ and $\vec{s}$ reduce to values, say $\vec{t} \rightsquigarrow^* \vec{v}$ and $\vec{s} \rightsquigarrow^* \vec{w}$ in at most a number $P_Q(\max(|\vec{t}|, |\vec{s}|))$ of reduction steps. Notice that, since $\vec{t}$ and $\vec{s}$ have type Q, in each step of the reduction they are represented by a superposition of at most $\dim(Q)$ terms, therefore the classical complexity of calculating the reduction differs only by a constant factor from the number of reduction steps. Finally, verifying that $\langle \vec{v} | \vec{w} \rangle = 0$ is also done in polynomial time since $|\vec{v}|, |\vec{w}| \leq P_Q(\max(|\vec{t}|, |\vec{s}|))$ and we only need to reduce at most a constant $\dim(Q)$ number of terms in superposition.
- ($\mathsf{ORTHO}_\times \in \mathsf{PTIME}$) Checking if a term contains free variables can be done in linear time on its size, i.e. in time $O(\max(|\vec{t}|, |\vec{s}|))$. If the terms are closed, computing $\mathsf{ORTHO}_\emptyset(\vec{t}, \vec{s})$ can be done in polynomial time. If the terms are open, we proceed inductively on the structure of the term until we find (or not) matching subterms that are closed and from which we can prove orthogonality. This can easily be done in polynomial time.
- ($\mathsf{ORTHO}_{\mathsf{untyped}} \in \Pi_1^0$) Since V_c is a countable set, it suffices to show that PERP is computable. To compute PERP, we only need to reduce the terms to values $\vec{w}_1$ and $\vec{w}_2$. Notice that, since the terms which we are using in the substitution may not have the correct type, this implies that the term we are reducing might not be typable in PUNQ, which renders Theorem 6.12 inapplicable and therefore we cannot place a (in particular, finite) bound on its number of reduction steps. However, by Theorem 6.12, we know that we need only apply a polynomial P_Q number of reduction steps and, if the term is not yet a value (i.e., by Theorem 3.4, it does not reduce), we know that it does not correspond to the candidate type Q in PUNQ, and therefore we do not need to consider it.
  If the terms do reduce to values $\vec{w}_1$ and $\vec{w}_2$, then we may check if they belong to $[\![Q]\!]_\emptyset$, the set of realizers of Q. Notice that, since the grammar of ground types Q does not include applications nor polymorphisms, we can check that a term belongs in $[\![Q]\!]_\emptyset$ purely syntactically. Finally, we can check $\mathsf{ORTHO}_\emptyset$ in polynomial time. □