A feasible and unitary programming language with quantum control

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

A feasible and unitary programming language with quantum control*

Alejandro Díaz-Caro[1,2], Emmanuel Hainry[3], Romain Péchoux[3], and Mário Silva[3]

[1] Universidad de Buenos Aires - CONICET, ICC, Buenos Aires, Argentina
[2] Universidad Nacional de Quilmes, DCyT, Bernal, PBA, Argentina
adiazcaro@conicet.gov.ar
[3] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy,France
{romain.pechoux,emmanuel.hainry,mmachado}@loria.fr

Abstract. We introduce PUNQ, a novel quantum programming language with quantum control, which features higher-order programs that can be superposed, enabling quantum control via quantum conditionals. 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. PUNQ seamlessly combines the two modalities: quantum circuit programs uphold unitarity, and all programs are evaluated in polynomial time, ensuring their feasibility.

1 Introduction

Motivations. *Classical control vs quantum control.* Quantum programming languages can be classified into two primary categories based on their control flow handling.

Firstly, *classical control* [41] 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 [42], Quipper [26], and Qwire [37]. 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 [45]. Notably, this category accommodates models like QRAM [33].

Conversely, *quantum control* [17] allows programming quantum operations based on quantum data, a fundamental concept in quantum computing. For

* 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.

example, the CNOT operation governs the application of a NOT operation based on a control qubit. More advanced examples like the Quantum Switch [39, 40] 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 quantum data. Programming languages like QML [3], Lambda-$\mathcal{S}_1$ [20], and Qunity [44] fall into this category. However, these programs must overcome significant constraints to ensure physical implementability, particularly in terms of efficient compilation into quantum circuits and 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 [3] to some extent, and more recently by [19], characterizing superpositions and isometries with realizability techniques [35, 43]. 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 [25, 30], has been introduced in [20] 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* [27] or *tractability* is equally crucial, considering practical problem-solving constraints like time, space, and resources. Compiling quantum programs to low-level models, like quantum circuits, necessitates 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 [47] links such families to Bounded-error Quantum Polynomial time (BQP) [12], a quantum analogue of BPP. Feasibility has deep roots in classical complexity, leading to fields like descriptive complexity [31] and implicit computational complexity [38] characterizing complexity classes logically and through programming languages.

Previous work [16] based on light linear logic [24] 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.

Contributions. We present PUNQ, a typed quantum programming language with quantum control addressing unitarity and feasibility. In PUNQ, programs are higher-order, allowing superposition. Quantum control employs a quantum conditional inspired by QML [3], producing superposed outputs from superposed inputs.

Unitarity is achieved via a variant of Lambda-$\mathcal{S}_1$ [20], introducing the $\sharp$ modality for superpositions, addressing quantum no-cloning. This modality marks non-duplicable types, treating superpositions of type $\sharp A$ linearly, with duplicable terms of types distinct from $\sharp A$. 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 Dual Light Affine Logic (DLAL) [8], employing the § modality to mark potential duplications. Notably, PUNQ differs from DLAL in two key aspects: i) PUNQ is not fully affine, in order to preserve unitarity, and ii) it employs a function ! on types (which is not a modality), mapping types to their duplicable counterparts, preserving the no-cloning property.

The main contributions of this paper are:

- a new typed programming language with quantum control that enjoys subject reduction (Theorem 3.5),
- 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 polynomial normalization result (Theorem 6.6), also ensuring polynomially bounded size of normal forms,
- a new notion of orthogonality that strictly extends the orthogonality of [20]. Though undecidable, we show that it is not more complicated than the non-halting problem and it is decidable in double-exponential time on the linear fragment of PUNQ with algebraic coefficients in $\overline{\mathbb{Q}}$ (Theorem 7.4),
- a complexity result for type inference over $\overline{\mathbb{Q}}$ (Theorem 7.5) that can be done in polynomial time, with an oracle computing orthogonality.

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 and 4.3), we provide the encoding of a simple quantum teleportation protocol (Example 4.4) illustrating that soundness can be used to certify unitarity, we also provide a quantum random walk algorithm (Example 4.5) illustrating polynomial time normalization.

Related work. *Quantum control and unitarity.* QML [3] 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 [5]. 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-$\mathcal{S}_1$ [19,20], as well as the current work, can be seen as the ℓ^2-norm preserving restriction of Lineal. Lambda-$\mathcal{S}$ [18,21], a preliminary version of this typing discipline, does not ensure unitarity but includes measurements. Qunity [44] enforces the no-cloning property using a non-standard notion of sharing, allowing duplicated variables to produce only entangled states. There have also been attempts [48] to define a notion of "quantum alternation" allowing measurements to be quantum-controlled, where the resulting system is not monotone with respect to the Löwner order and, hence, cannot be considered to be a physically feasible concept [7]. An alternative approach in [49] 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 [13,14]. 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 [38] for a survey). To mention a few of them, [10] provided the first implicit (i.e., where the complexity bound does not need to be explicited by the programmer) characterization of polynomial time and [22] 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 [16] characterizes BQP on the quantum lambda-calculus. However, due to the presence of unrestricted measurement, this classically-controlled system cannot guarantee unitarity. Finally, [28,46] provide two characterizations of FBQP with quantum control which are restricted to first-order.

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 1. A *term* can take the form of a variable x, a bit $|0\rangle$ or $|1\rangle$, a conditional statement, an abstraction, an application, a pair, or a pair destructor. We denote the set of terms as $\mathtt{T}$, and

$$t := x \mid |0\rangle \mid |1\rangle \mid \mathsf{if}\ t\ \mathsf{then}\ \vec{t}\ \mathsf{else}\ \vec{t} \mid \lambda x.\vec{t} \mid t\ t \mid (t,t) \mid \mathsf{let}\ \ (x,y) = t\ \mathsf{in}\ \vec{t} \qquad (\mathtt{T})$$
$$\vec{t} := t \mid \vec{0} \mid \alpha \cdot \vec{t} \mid \vec{t} + \vec{t} \qquad (\mathtt{S})$$
$$v := |0\rangle \mid |1\rangle \mid \lambda x.\vec{t} \mid (v,v) \qquad (\mathtt{BV})$$
$$\vec{v} := v \mid \vec{0} \mid \alpha \cdot \vec{v} \mid \vec{v} + \vec{v} \qquad (\mathtt{V})$$

Fig. 1. Syntax of PUNQ programs.

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 a polytime computable complex number $\alpha \in \tilde{\mathbb{C}}$ [2], or the sum $\vec{t_1} + \vec{t_2}$ of two superpositions. We represent the set of superpositions as $\mathtt{S}$, 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 $\mathtt{BV}$ and $\mathtt{V}$ 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 $\mathtt{S}_c$ 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,

$$|x| \triangleq 1 \qquad |t_1\ t_2| = |(t_1,t_2)| \triangleq |t_1| + |t_2| + 1$$
$$||0\rangle| \triangleq 1 \qquad |\mathsf{let}\ \ (x,y) = t\ \mathsf{in}\ \vec{t}| \triangleq 1 + |t| + |\vec{t}|$$
$$||1\rangle| \triangleq 1 \qquad |\vec{0}| \triangleq 0$$
$$|\mathsf{if}\ t\ \mathsf{then}\ \vec{t_1}\ \mathsf{else}\ \vec{t_2}| \triangleq 1 + |t| + \max(|\vec{t_1}|, |\vec{t_2}|) \qquad |\alpha \cdot \vec{t}| \triangleq |\vec{t}|$$
$$|\lambda x.\vec{t}| \triangleq 1 + |\vec{t}| \qquad |\vec{t_1} + \vec{t_2}| \triangleq \max(|\vec{t_1}|, |\vec{t_2}|)$$

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

$$\vec{t_1} + \vec{t_2} \equiv \vec{t_2} + \vec{t_1} \qquad (\vec{t_1} + \vec{t_2}) + \vec{t_3} \equiv \vec{t_1} + (\vec{t_2} + \vec{t_3})$$
$$\vec{0} + \vec{t} \equiv \vec{t} \qquad 0 \cdot \vec{t} \equiv \vec{0}$$
$$1 \cdot \vec{t} \equiv \vec{t} \qquad \alpha \cdot (\beta \cdot \vec{t}) \equiv \alpha\beta \cdot \vec{t}$$
$$\alpha \cdot \vec{t} + \beta \cdot \vec{t} \equiv (\alpha + \beta) \cdot \vec{t} \qquad \alpha \cdot (\vec{t_1} + \vec{t_2}) \equiv \alpha \cdot \vec{t_1} + \alpha \cdot \vec{t_2}$$

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

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

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

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

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

Fig. 2. Syntactic sugar on PUNQ syntax.

$\forall i \neq j, t_i \neq t_j$, 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 2.

2.2 Operational semantics

The semantics of PUNQ programs is defined by the rewrite relation $\rightsquigarrow \subseteq \mathtt{S}_c \times \mathtt{S}_c$, given in Figure 3. We denote its reflexive and transitive closure as $\rightsquigarrow^*$.

It is important to emphasize that the defined syntactic sugar above is used in the reduction rules (If_+), (App), ($\mathrm{App}_{\mathtt{V}}$), (Pair), ($\mathrm{Pair}_{\mathtt{V}}$), and ($\mathrm{Let}_+$) to simplify notations. For instance, the reduction of rule (If_+) can be written as

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

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 $t \in \mathtt{T},\ v \in \mathtt{V}$, 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);

$$\frac{}{\mathsf{if}\ |0\rangle\ \mathsf{then}\ \vec{s}\ \mathsf{else}\ \vec{t} \rightsquigarrow \vec{s}}\ (\mathrm{If}_0) \qquad \frac{}{\mathsf{if}\ |1\rangle\ \mathsf{then}\ \vec{s}\ \mathsf{else}\ \vec{t} \rightsquigarrow \vec{t}}\ (\mathrm{If}_1)$$

$$\frac{}{(\lambda x.\vec{t})\ v \rightsquigarrow \vec{t}[v/x]}\ (\mathrm{Abs}) \qquad \frac{}{\mathsf{let}\ (x,y) = (v,w)\ \mathsf{in}\ \vec{t} \rightsquigarrow \vec{t}[v/x, w/y]}\ (\mathrm{Let})$$

$$\frac{t \rightsquigarrow \vec{s}}{\mathsf{if}\ t\ \mathsf{then}\ \vec{r}_1\ \mathsf{else}\ \vec{r}_2 \rightsquigarrow \mathsf{if}\ \vec{s}\ \mathsf{then}\ \vec{r}_1\ \mathsf{else}\ \vec{r}_2}\ (\mathrm{If}_+)$$

$$\frac{t \rightsquigarrow \vec{s}}{r\ t \rightsquigarrow r\ \vec{s}}\ (\mathrm{App}) \quad \frac{t \rightsquigarrow \vec{s}}{t\ v \rightsquigarrow \vec{s}\ v}\ (\mathrm{App_v}) \quad \frac{t \rightsquigarrow \vec{s}}{(t,r) \rightsquigarrow (\vec{s},r)}\ (\mathrm{Pair}) \quad \frac{t \rightsquigarrow \vec{s}}{(v,t) \rightsquigarrow (v,\vec{s})}\ (\mathrm{Pair_v})$$

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

$$\frac{\sum_{i\in I} \alpha_i \cdot t_i + \sum_{j\in J} \beta_j \cdot v_j \in \mathrm{CF} \qquad \forall i \in I,\ t_i \rightsquigarrow s_i}{\sum_{i\in I} \alpha_i \cdot t_i + \sum_{j\in J} \beta_j \cdot v_j \rightsquigarrow \sum_{i\in I} \alpha_i \cdot s_i + \sum_{j\in J} \beta_j \cdot v_j}\ (\mathrm{Sup})$$

$$\frac{\vec{t} \equiv \vec{t}_1 \qquad \vec{t}_1 \rightsquigarrow \vec{s}_1 \qquad \vec{s}_1 \equiv \vec{s}}{\vec{t} \rightsquigarrow \vec{s}}\ (\mathrm{Equ})$$

Fig. 3. Semantics of PUNQ programs.

– 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-$\mathcal{S}_1$ [20] together with the polytime strong normalization properties of the DLAL [8] type system.

3.1 Types, judgments, and environments

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

$$A := X \mid \mathbb{B} \mid A \multimap A \mid A \Rightarrow A \mid A \times A \mid \sharp A \mid \S A \mid \forall X.A.$$

We use A, B, C, and so on for 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 $\mathbb{T}_c$. Intuitively, objects of type $\sharp A$ are unitary superpositions of elements of type A 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 $\alpha, \beta \in \tilde{\mathbb{C}}$ 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 possible qubit duplication or erasure.

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 $\langle -|-\rangle : \mathtt{V}_c \times \mathtt{V}_c \to \tilde{\mathbb{C}}$ over closed values as

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

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 $\mathtt{V}_c$ is a Hilbert space as $\ell^2(\mathtt{V}_c) = \mathtt{V}_c$.

In order to preserve unitarity, the type system will use the following notion of orthogonality to ensure that superpositions or conditionals guarded by a superposition preserve the ℓ^2 norm.

Definition 3.1 (Orthogonality). *The* orthogonality relation $\perp_A \subseteq \mathtt{S}_c \times \mathtt{S}_c$*, for a type* $A \in \mathbb{T}_c$ *is defined by* $\vec{t} \perp_A \vec{s}$ *iff* $F_A(\vec{t}, \vec{s}) = 0$ *where* $F_A(-,-) : \mathtt{S}_c \times \mathtt{S}_c \to \tilde{\mathbb{C}}$ *is defined inductively on the closed type* A *by the rules of Figure 4.*

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 $\vec{t},\ \vec{s} \in \mathtt{S}$ 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 $\forall \sigma_{\Gamma \cup \Delta},\ \vec{t}\sigma_{\Gamma\cup\Delta} \perp_A \vec{s}\sigma_{\Gamma\cup\Delta}$.

Notice that checking orthogonality requires information on the type of the term. However, it is important to highlight that orthogonality checks and type checking can be carried out independently, as detailed in Theorem 7.3.

Intuitively, two closed values of type $\sharp\mathbb{B}$ are orthogonal if their inner product is 0. Two pairs are orthogonal if at least one of their projections reduce to orthogonal values. Finally, two superpositions of functional type are orthogonal if when applied to any closed value, they reduce to orthogonal values.

Example 3.2 (Orthogonality of values). Let $|\pm\rangle \triangleq \frac{1}{\sqrt{2}} \cdot (|0\rangle \pm |1\rangle)$. Then $|+\rangle \perp_{\sharp\mathbb{B}} |-\rangle$ (both have type $\sharp\mathbb{B}$, see Example 4.1) as $F_{\sharp\mathbb{B}}(|+\rangle, |-\rangle) = \langle +|-\rangle = 0$.

Example 3.3 (Orthogonality of functions). The terms $\mathtt{f} \triangleq \lambda x.\, (|0\rangle, x)$ and $\mathtt{g} \triangleq \lambda x.\, (|1\rangle, x)$, both having the type $\sharp\mathbb{B} \multimap \sharp(\mathbb{B} \times \mathbb{B})$, satisfy $F_{\sharp\mathbb{B} \multimap \sharp(\mathbb{B}\times\mathbb{B})}(\mathtt{f}, \mathtt{g}) = 0$.

$$
\begin{aligned}
F_{\mathbb{B}}(\vec{t},\vec{s}) &\triangleq \langle \vec{v}|\vec{w}\rangle, && \text{if } \vec{t} \rightsquigarrow^* \vec{v} \text{ and } \vec{s} \rightsquigarrow^* \vec{w},\\
F_{A\multimap B}(\vec{t},\vec{s}) &\triangleq 0, && \text{if } \forall v \in \mathtt{BV}_A,\ F_B(\vec{t}\ v, \vec{r}\ v) = 0,\\
F_{A\Rightarrow B}(\vec{t},\vec{s}) &\triangleq 0, && \text{if } \forall v \in \mathtt{BV}_A,\ F_B(\vec{t}\ v, \vec{r}\ v) = 0,\\
F_{A\times B}(\vec{t},\vec{s}) &\triangleq F_A(\vec{v}_1,\vec{w}_1).F_B(\vec{v}_2,\vec{w}_2), && \text{if } \vec{t} \rightsquigarrow^* \langle \vec{v}_1,\vec{v}_2\rangle \text{ and } \vec{s} \rightsquigarrow^* \langle \vec{w}_1,\vec{w}_2\rangle,\\
F_{\sharp A}(\vec{t},\vec{s}) &\triangleq \sum_{i,j} \overline{\alpha_i}\beta_j.F_A(\vec{v}_i,\vec{w}_j), && \text{if } \vec{t} \rightsquigarrow^* \sum_i \alpha_i \cdot \vec{v}_i \text{ and } \vec{s} \rightsquigarrow^* \sum_j \beta_j \cdot \vec{w}_j,\\
F_{\S A}(\vec{t},\vec{s}) &\triangleq F_A(\vec{t},\vec{s}), &&\\
F_{\forall X.A}(\vec{t},\vec{s}) &\triangleq 0, && \text{if } \forall B \in \mathbb{T}_c,\ F_{A[B/X]}(\vec{t},\vec{s}) = 0,\\
F_A(\vec{t},\vec{s}) &\triangleq 1, && \text{in any other case.}
\end{aligned}
$$

Fig. 4. Orthogonality function.

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

$$
\begin{aligned}
!(X) &\triangleq X & !(\mathbb{B}) &\triangleq \mathbb{B}\\
!(A \multimap B) &\triangleq A \multimap B & !(A \Rightarrow B) &\triangleq A \Rightarrow B\\
!(A \times B) &\triangleq !(A) \times !(B) & !(\sharp A) &\triangleq !(A)\\
!(\S A) &\triangleq \S!(A) & !(\forall X.A) &\triangleq \forall X.!(A)
\end{aligned}
$$

This function is reminiscent of the bang modality in linear logic [23] 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. On the other hand, $!(\sharp(\sharp\mathbb{B} \multimap \sharp\mathbb{B})) = \sharp\mathbb{B} \multimap \sharp\mathbb{B}$ transforms a non-clonable type (the superpositions of unitary maps over qubits) into a clonable type (unitary maps over qubits).

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 A is considered as the base of a vector space for its values, then $\sharp A$ 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 $A \leq \sharp A$ and $\sharp A = \sharp\sharp A$. Additionally, it can be shown that $\sharp A \times \sharp B \leq \sharp(!(A) \times !(B))$, 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 $\sharp(\mathbb{B} \times \mathbb{B})$.

$$\frac{}{A \leq A} \qquad \frac{}{A \leq \sharp A} \qquad \frac{}{\sharp\sharp A \leq \sharp A} \qquad \frac{}{A \leq \sharp!(A)} \qquad \frac{}{\sharp\S A \leq \S\sharp A}$$

$$\frac{A \leq B \quad B \leq C}{A \leq C} \qquad \frac{A' \leq A \quad B \leq B'}{A \multimap B \leq A' \multimap B'} \qquad \frac{A' \leq A \quad B \leq B'}{A \Rightarrow B \leq A' \Rightarrow B'}$$

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

Fig. 5. Subtyping relation.

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:

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

where $k \in \{+, -\}$ and where ! is the bang function from Definition 3.4. 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. Several key points are worth highlighting:

- The ! function is employed in rule ($\Rightarrow_i$) to ensure the no-cloning property.
- 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 ($\Rightarrow_i$).
- Rules ($\mathsf{if}_\sharp$) and ($\sharp_i$) are the only two rules that make use of the orthogonality predicate $\perp_A \subseteq \mathbb{T}_c \times \mathbb{T}_c$.
- Rule ($\forall_e$) utilizes the type substitution $[B/X]$.

PUNQ is the set of typable closed superpositions in $\mathtt{S}_c$.
Proving subject reduction for PUNQ programs is straightforward.

Theorem 3.5 (Subject reduction). *If* $;\vdash \vec{t} : A$ *and* $\vec{t} \rightsquigarrow \vec{r}$*, then* $;\vdash \vec{r} : A$.

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

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

$$\frac{\Gamma;\Delta\vdash t:\mathbb{B} \qquad \forall i,\ \Gamma';\Delta'\vdash\vec{s_i}:A}{\Gamma,\Gamma';\Delta,\Delta'\vdash\mathsf{if}\ t\ \mathsf{then}\ \vec{s}_1\ \mathsf{else}\ \vec{s}_2:A}\ (\mathsf{if})$$

$$\frac{\Gamma;\Delta\vdash t:\sharp\mathbb{B} \qquad \forall i,\ \Gamma';\Delta'\vdash\vec{s}_i:A \qquad \vec{s}_1\perp_A^{\Gamma';\Delta'}\vec{s}_2}{\Gamma,\Gamma';\Delta,\Delta'\vdash\mathsf{if}\ t\ \mathsf{then}\ \vec{s}_1\ \mathsf{else}\ \vec{s}_2:\sharp A}\ (\mathsf{if}_\sharp)$$

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

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

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

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

$$\frac{\forall i,\ \Gamma;\Delta\vdash\vec{t}_i:A \qquad \forall j\neq k,\ \vec{t}_j\perp_A^{\Gamma;\Delta}\vec{t}_k \qquad \sum_{i=1}^n|\alpha_i|^2=1}{\Gamma;\Delta\vdash\sum_{i=1}^n\alpha_i\cdot\vec{t}_i:\sharp A}\ (\sharp_i)$$

$$\frac{;\Gamma,\Delta\vdash\vec{t}:A}{\Gamma;\S\Delta\vdash\vec{t}:\S A}\ (\S_i) \qquad \frac{\Gamma;\Delta\vdash s:\S B \qquad \Gamma';\Delta',x:\S B\vdash t:A}{\Gamma,\Gamma';\Delta,\Delta'\vdash t[s/x]:A}\ (\S_e)$$

$$\frac{\Gamma;\Delta\vdash\vec{t}:A \qquad X\notin FV(\Gamma,\Delta)}{\Gamma;\Delta\vdash\vec{t}:\forall X.A}\ (\forall_i) \qquad \frac{\Gamma;\Delta\vdash\vec{t}:\forall X.A}{\Gamma;\Delta\vdash\vec{t}:A[B/X]}\ (\forall_e)$$

Fig. 6. Typing rules for PUNQ programs.

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

3.4 Intuitions

Most rules in Figure 6 are reminiscent of the typing discipline of Dual Light Affine Logic [8], 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 without breaking unitarity is impossible. 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 ${}^{\mathrm{s}_c}/_{\equiv} \times {}^{\mathrm{s}_c}/_{\equiv}$ 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 $(\mathsf{if}_\sharp)$ 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 3. Hence, the result has type $\sharp A$.

The typing rules for linear and intuitionistic arrows resemble those in [9], with the additional requirement that the introduction and elimination of the intuitionistic arrow, 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 [8]. 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 $\sharp A$ and $\sharp B$ to preserve unitarity. Note that this typing rule can be extended to superpositions as follows:

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

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 [8] 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 [24]) 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 $\perp_A$, and that the norm of the generated superposition $\sum_{i=1}^{n} |\alpha_i|^2$ equals 1. 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 will be denoted $\tilde{\mathbb{C}}$ 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 [2, 12].

Our notion of orthogonality is more general than that in [20], where the superposition of functions is disallowed. It is also treated more realistically than in [19], where the superposition of functions is generally not a function. In our setting, a superposition of functions $\sharp(A \multimap B)$ can be coerced to a function in $A \multimap \sharp B$. It is important to note that the predicate $\perp_A$ requires checking some normalization properties on superpositions (see Figure 4) and is generally not decidable. While this might be seen as a stringent requirement by the type system, it is not a limitation for practical reasons. In rules $(\mathsf{if}_\sharp)$ and $(\sharp_i)$ of Figure 6, the predicate applies only to typable superpositions. As we will demonstrate (Theorem 6.6), these superpositions normalize in polynomial time on each value, making their reduction decidable in polynomial time. The challenge in orthogonality proofs is closely tied to the use of functional types, which requires checking for each possible value. Options to regain decidability include disallowing orthogonality on functions, as in [20], or treating orthogonality proofs as certificates that can be delegated to an external prover or oracle. In the next section, we provide examples of how to encode basic operators in quantum computing, all of which rely solely on the orthogonality predicate in values, a property that is decidable.

4 Examples

We consider several simple examples to further motivate the typed language PUNQ.

Example 4.1 (X-basis states). The one-qubit states $|\pm\rangle \triangleq \frac{1}{\sqrt{2}} \cdot (|0\rangle \pm |1\rangle)$ have type $\sharp\mathbb{B}$.

$$\pi_\pm \triangleq \dfrac{\dfrac{\dfrac{}{\vdash |0\rangle : \mathbb{B}}\,(0) \quad \dfrac{}{\vdash |1\rangle : \mathbb{B}}\,(1) \quad |0\rangle \perp_{\mathbb{B}} |1\rangle \quad \left|\frac{1}{\sqrt{2}}\right|^2 + \left|\pm\frac{1}{\sqrt{2}}\right|^2 = 1}{\vdash \frac{1}{\sqrt{2}} \cdot |0\rangle \pm \frac{1}{\sqrt{2}} \cdot |1\rangle : \sharp\mathbb{B}}\,(\sharp_i)}{\vdash |\pm\rangle : \sharp\mathbb{B}}\,(\equiv)$$

Gates are also easy to derive using the quantum control. The unitarity of the gates guarantees that any branches are orthonormal, and so they will fit the conditions of $\mathsf{if}_\sharp$.

Example 4.2 (Hadamard and Z gates). The term $\mathsf{H} \triangleq \lambda x.\mathsf{if}\ x\ \mathsf{then}\ |+\rangle\ \mathsf{else}\ |-\rangle$ can be derived with type $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$:

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

where $|+\rangle \perp_{\sharp\mathbb{B}} |-\rangle$ is shown in Example 3.2. The term $\mathsf{Z} \triangleq \lambda x.\mathsf{if}\ x\ \mathsf{then}\ |0\rangle\ \mathsf{else}\ -1\cdot|1\rangle$ is also derivable with the same type:

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

The syntax also lends itself well to construction of quantum controlled gates.

Example 4.3 (Controlled NOT gate). Let $\mathsf{NOT} \triangleq \lambda x.\mathsf{if}\ x\ \mathsf{then}\ |1\rangle\ \mathsf{else}\ |0\rangle$ which can be derived with type $\sharp\mathbb{B} \multimap \sharp\mathbb{B}$. We can derive a term representing the controlled gate $\mathsf{CNOT} \triangleq \lambda z.\mathsf{let}\ (x,y) = z\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ (|0\rangle, y)\ \mathsf{else}\ (|1\rangle, \mathsf{NOT}\ y)$ with type $\sharp(\mathbb{B}\times\mathbb{B}) \multimap \sharp(\mathbb{B}\times\mathbb{B})$.

In the previous example, even though $|0\rangle$ and $|1\rangle$ represent the state of x (the control qubit) in each branch of CNOT, the 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.

Example 4.4 (Quantum teleportation). As a more evolved example illustrating PUNQ's expressive power, let us examine the case of quantum teleportation [11] with delayed measurements.

We will use $|ab\rangle$, where $a, b \in \{0,1\}$, as a shorthand for basis states $(|a\rangle, |b\rangle)$ in a two-qubit vector space of type $\sharp(\mathbb{B}^2)$. For $n \geq 1$, let $\mathbb{B}^{n+1} \triangleq \mathbb{B}\times\mathbb{B}^n$ and $\mathbb{B}^1 \triangleq \mathbb{B}$. The term $\mathsf{Bell} \triangleq \lambda z.\mathsf{let}\ (x,y) = z\ \mathsf{in}\ \mathsf{CNOT}\,(\mathsf{H}\ x, y)$ with type $\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2)$, represents the circuit that computes the map sending $|ab\rangle$ to $\frac{1}{\sqrt{2}}(|0b\rangle + (-1)^a\cdot|1(1-b)\rangle)$. 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, considering delayed measurements, with the following terms:

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

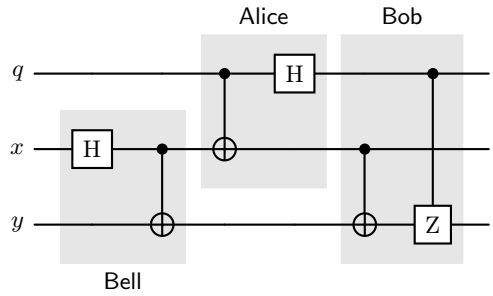

Fig. 7. Circuit for quantum teleportation with delayed measurements.

These terms can be typed by $\sharp(\mathbb{B}^2) \multimap \sharp(\mathbb{B}^2)$ and $\sharp(\mathbb{B}^3) \multimap \sharp(\mathbb{B}^3)$, respectively. Then, $\mathsf{telep} \triangleq \lambda z.\mathsf{let}\ (q,w) = z\ \mathsf{in}\ (\mathsf{let}\ (x,y) = \mathsf{Bell}\ w\ \mathsf{in}\ \mathsf{Bob}\ (\mathsf{Alice}\ (q,x)\,,y))$ can be derived with type $\sharp(\mathbb{B}^3) \multimap \sharp(\mathbb{B}^3)$, encoding quantum teleportation with delayed measurements, depicted in Figure 7.

Example 4.5 (Quantum random walk). Consider the quantum random walk [15, 32], the quantum analogue of Markov chains, as an illustrating example for polytime iteration. This algorithm is known to provide a quantum speedup over its classical analog in different applications [4] and the corresponding circuit is provided in Figure 8. To keep the example simple, we will consider a walk on a loop of k nodes, although our reasoning can be adapted to more general cases. We define the R and L operators, for $x_1, \ldots, x_k \in \{0,1\}$, as

$$\mathsf{R}\ |x_1 \ldots x_k\rangle \mapsto |x_k\, x_1 \ldots x_{k-1}\rangle \quad \text{and} \quad \mathsf{L}\ |x_1 \ldots x_k\rangle \mapsto |x_2 \ldots x_k\, x_1\rangle.$$

These correspond to unitary operators in the space $\tilde{\mathbb{C}}^{2^k} \to \tilde{\mathbb{C}}^{2^k}$, and therefore by Theorem 5.7 we show that there exist corresponding PUNQ terms 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:

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

where x corresponds to the *coin* qubit deciding the direction, on which we continuously apply the Hadamard gate. We can now iterate this term using DLAL Church numerals of type $\mathbb{N}$ and inhabitants $\underline{n}$, where

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

For any polynomial P of degree 2^d, there is a DLAL term $\mathsf{poly} : \mathbb{N} \multimap \S^{2d}\mathbb{N}$ that simulates it [8, Proposition 9]. Using this term we can compose step a polynomial number of times:

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

such that $\mathsf{walk}\ \underline{n} \rightsquigarrow^* \lambda x.\mathsf{step}(\mathsf{step}\ (\ldots(\mathsf{step}\ x)\ldots) : \sharp(\mathbb{B}^{k+1}) \multimap \S^{2^d+1}\sharp(\mathbb{B}^{k+1})$, i.e., the composition $P(n)$ times of a single step.

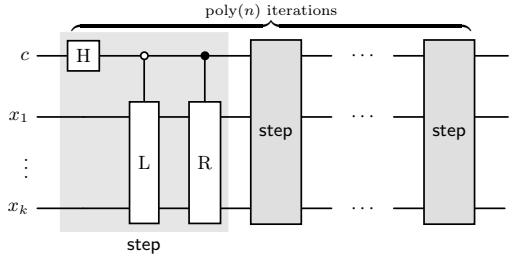

Fig. 8. Circuit for polytime quantum random walk.

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 in [20] employs the realizability techniques developed in [19], 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 $\Rightarrow$, 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 $\vec{v} \in \mathtt{V}_c$ is defined as the set of its basis values:

$$\begin{aligned} \mathrm{base}(v) &\triangleq \{v\} & \mathrm{base}(\alpha \cdot \vec{v}) &\triangleq \mathrm{base}(\vec{v}) \\ \mathrm{base}(\vec{0}) &\triangleq \emptyset & \mathrm{base}(\vec{v_1} + \vec{v_2}) &\triangleq \mathrm{base}(\vec{v_1}) \cup \mathrm{base}(\vec{v_2}) \end{aligned}$$

Given a set of closed values S in a vector space $\mathtt{V}_c$, the *span* (span) and *basis* ($\flat$) of S are defined as follows:

$$\begin{aligned} \mathsf{span}(S) &\triangleq \Big\{ \sum_{i=1}^{n} \alpha_i \cdot \vec{v_i} : n \geq 0, \alpha_1, \ldots, \ \alpha_n \in \tilde{\mathbb{C}}, \ \vec{v_1}, \ldots, \vec{v_n} \in S \Big\} & &\subseteq \mathtt{V}_c \\ \flat S &\triangleq \bigcup_{\vec{v} \in S} \mathrm{base}(\vec{v}) & &\subseteq \mathtt{BV}_c \end{aligned}$$

We first define the *realizability predicate* for a superposition $\vec{t}$ and set of closed values $S \subseteq \mathtt{V}_c$, 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 $\{\vec{t} \in \mathtt{S}_c : \vec{t} \Vdash S\}$.

Let $\mathcal{S}_1 \subseteq \mathtt{V}_c$ be the set of unit values, that is, $\mathcal{S}_1 \triangleq \{\vec{v} \in \mathtt{V}_c : |\langle \vec{v} | \vec{v} \rangle|^2 = 1\}$. 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.

The following property of the definition of ! (Definition 3.4) clarifies the nature of $\flat$ and the role of ! in erasing superposition.

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

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

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

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

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 $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$ is* isometric *if it preserves the inner product, i.e., if $\langle F(u)|F(v)\rangle = \langle u|v\rangle$, $\forall u, v \in \tilde{\mathbb{C}}^{2^n}$. Furthermore, if $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^n}$ is isometric then F is* unitary.

To simplify notation, let $|i\rangle$ for $i = 0, \ldots, 2^n - 1$ represent the n-qubit term $(|i_1\rangle, (|i_2\rangle, \ldots (|i_{n-1}\rangle, |i_n\rangle) \ldots))$ in the binary expansion of i. We show that for unitarity to hold, transformations between qubit terms must preserve the orthogonality and the norm of basis states.

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

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

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 $(\!|-|\!)$ be a map from closed values of type $\sharp(\mathbb{B}^n)$ to $\tilde{\mathbb{C}}^{2^n}$ defined by $(\!|\sum_{i=0}^{2^n-1} \alpha_i \cdot |i\rangle|\!) \triangleq \begin{pmatrix} \alpha_0 & \cdots & \alpha_{2^n-1} \end{pmatrix}^T \in \tilde{\mathbb{C}}^{2^n}$. A closed superposition $\vec{t} \in \mathrm{PUNQ}$ of type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ *represents* an operator $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$ 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 $F((\!|\vec{v}|\!)) = (\!|\vec{w}|\!)$.

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 match the abstraction represents a unitary operator.

Theorem 5.6 (Soundness). *For any closed superposition* $\vec{t} \in \mathrm{PUNQ}$ *of type* $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ *and any* $k, n \in \mathbb{N} - \{0\}$*, the following hold:*

1. *If* $k \geq n$*, then* $\vec{t}$ *represents an isometry in* $\tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$*, and*
2. *If* $k = n$*, then* $\vec{t}$ *represents a unitary operator in* $\tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^n}$*.*

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 $\vec{t}[x/|i\rangle] \rightsquigarrow^* \vec{v}_i$ with $\langle \vec{v}_i | \vec{v}_j \rangle = 0$, for $i \neq j$. Let $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$ be the linear operator defined by $F((\!||i\rangle|\!)) = (\!|\vec{v}_i|\!)$, for each $i = 0, \ldots, 2^n - 1$. By the linearity of the calculus we have that $\lambda x.\vec{t}$ represents the operator F. Moreover, F is isometric by Definition 5.4 since $\|(\!|\vec{v}_i|\!)\| = 1$ and $(\!|\vec{v}_i|\!)^\dagger \cdot (\!|\vec{v}_j|\!) = 0$, 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 absurdity. 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 $\beta_i \in \tilde{\mathbb{C}}$ such that $\sum_i \beta_i \cdot \vec{v}_i = \vec{0}$ 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. □

Theorem 5.7 (Completeness). *For any isometry $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$, with $k \geq n \geq 1$, there exists a closed superposition $\vec{t} \in \mathtt{S}_c$ of type $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)$ that represents F.*

Proof. If $\lambda x.\vec{t}$ represents an isometric operator $F : \tilde{\mathbb{C}}^{2^n} \to \tilde{\mathbb{C}}^{2^k}$ then $(\lambda x.\vec{t})|i\rangle \rightsquigarrow^* \vec{v_i}$ for some $\vec{v_i} \in \mathsf{span}(|0\rangle, |1\rangle, \ldots, |2^k - 1\rangle)$ such that $(\!|\vec{v_i}|\!) = F((\!|i|\!))$. We have that $\vec{t}[x/|i\rangle] \rightsquigarrow^* \vec{v_i} \in [\![\sharp(\mathbb{B}^k)]\!]_\emptyset$, for each $i = 0, \ldots, 2^n - 1$, since $\|\vec{v_i}\| = \|F((\!|i|\!))\| = 1$. From Lemma 5.5 we deduce that $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^k)]\!]_\emptyset$, since it holds that, for $i \neq j$, $\langle \vec{v_i} | \vec{v_j} \rangle = (\!|i|\!)^\dagger \cdot (\!|j|\!) = 0$. □

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 $n, k \in \mathbb{N} - \{0\}$, such that $n > k$, the type $\sharp(\mathbb{B}^n) \multimap \left(\sharp(\mathbb{B}^{n-k}) \times \sharp(\mathbb{B}^k)\right)$ is uninhabited.*

Proof. Consider an argument by absurdity, starting with a candidate closed term $\lambda x.\vec{t} \in [\![\sharp(\mathbb{B}^n) \multimap \left(\sharp(\mathbb{B}^{n-k}) \times \sharp(\mathbb{B}^k)\right)]\!]_\emptyset$. Define the values $(\vec{u_i}, \vec{v_i}) \in [\![\sharp(\mathbb{B}^{n-k}) \times \sharp(\mathbb{B}^k)]\!]_\emptyset$ satisfying $(\lambda x.\vec{t})\ |i\rangle \rightsquigarrow^* (\vec{u_i}, \vec{v_i})$. Notice that $\lambda x.\vec{t}$ can be subtyped as $\sharp(\mathbb{B}^n) \multimap \sharp(\mathbb{B}^n)$ and therefore by Theorem 5.6 must satisfy unitarity. We conclude, therefore, that since we have 2^n 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 $[\![\sharp(\mathbb{B}^n)]\!]_\emptyset$. 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 $\vec{s} \notin [\![\sharp(\mathbb{B}^{n-k}) \times \sharp(\mathbb{B}^k)]\!]_\emptyset$. However, we have that $\sum_i \alpha_i \cdot |i\rangle \in [\![\sharp(\mathbb{B}^n)]\!]_\emptyset$ and that $\lambda x.\vec{t}\ (\sum_i \alpha_i \cdot |i\rangle) \rightsquigarrow^* \vec{s}$. □

6 Polytime normalization

In this section, we show that PUNQ preserves the polytime strong normalization properties of DLAL [8]. For that purpose, we define an encoding from superpositions in PUNQ to a set of DLAL terms simulating this program and shows that the polynomial time strong normalization of each term 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.

6.1 Dual light affine logic

DLAL terms [8] are a strict subset of System F, 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 $\Gamma; \Delta \vdash_{\mathrm{DLAL}} \mathcal{S} : A$ if $\forall t \in \mathcal{S}$, $\Gamma; \Delta \vdash t : A$ can be derived in DLAL.

$$\frac{\Gamma;\Delta \vdash t : A}{\Gamma,\Gamma';\Delta,\Delta' \vdash t : A} \qquad \frac{\Gamma, x : B, y : B;\Delta \vdash t : A}{\Gamma, x : B;\Delta \vdash t[x/y] : A} \qquad \frac{}{;x : A \vdash x : A}$$

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

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

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

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

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

The size of a term t in DLAL, 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., $|\mathcal{S}| \triangleq \max\{|t| \mid t \in \mathcal{S}\}$.

Theorem 6.1 (Polynomial time strong normalization [8, 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.* □

The above theorem implies both a polynomial bound on the reduction length and a polynomial bound on the size of each term obtained during the reduction.

Given two sets of terms $\mathcal{S}$ and $\mathcal{T}$, we write $\mathcal{S} \to \mathcal{T}$ if $\mathcal{T} = \{t \mid s \in \mathcal{S} \wedge s \to t\}$. The set $\mathcal{T}$ is obtained by reducing the non-normalized terms of $\mathcal{S}$. We establish that $\emptyset$ does not reduce ($\emptyset \not\to$). As an example, consider the reductions

$$\big\{(\lambda f.\lambda x.x)(\lambda y.y), (\lambda x.\lambda y.x)\ u\ v\big\} \to \big\{\lambda x.x, (\lambda y.u)\ v\big\} \to \big\{u\big\} \to \emptyset.$$

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 $\mathcal{S} \to^k \mathcal{T}$. Note that $\mathcal{S} \to^{\leq n} \emptyset$ means that all terms in $\mathcal{S}$ reduce to their normal form 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 $\Gamma;\Delta \vdash_{\text{DLAL}} \mathcal{S} : A$, then $\mathcal{S} \to^{\leq P_A(|\mathcal{S}|)} \emptyset$ and, for each $\mathcal{T}$ satisfying $\mathcal{S} \to^n \mathcal{T}$, $|\mathcal{T}| \leq P_A(|\mathcal{S}|)$.*

Notice that the polynomial P_A can be explicitly computed and that the degree only depends on A as the depth of a term is a function of its type (see [8]).

$$[x] \triangleq \{x\} \qquad [|0\rangle] \triangleq \{\lambda x.\lambda y.x\} \qquad [|1\rangle] \triangleq \{\lambda x.\lambda y.y\}$$

$$[\mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2] \triangleq ([t]\ [t_1]\ \{*\}) \cup ([t]\ \{*\}\ [t_2])$$

$$[\lambda x.\vec{t}] \triangleq \{\lambda x.u \mid u \in [\vec{t}]\} \qquad [t_1\ t_2] \triangleq [t_1]\ [t_2]$$

$$[(t_1, t_2)] \triangleq \{\lambda x.x\ u_1\ u_2 \mid u_1 \in [t_1],\ u_2 \in [t_2]\}$$

$$[\mathsf{let}\ (x, y) = t_1\ \mathsf{in}\ t_2] \triangleq [t_1]\ \{(\lambda x.\lambda y.u_2) \mid u_2 \in [t_2]\}$$

$$[\vec{0}] \triangleq \{*\} \qquad [\alpha \cdot \vec{t}] \triangleq [\vec{t}] \qquad [\vec{t}_1 + \vec{t}_2] \triangleq [\vec{t}_1] \cup [\vec{t}_2]$$

Fig. 11. $[\cdot] : \mathrm{PUNQ} \to \mathcal{P}(\mathrm{DLAL}_*)$

6.2 Encoding of PUNQ in DLAL

The encoding from PUNQ to DLAL is standard except for the conditional $\mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$. In DLAL, a term encoding this conditional would be typed multiplicatively, requiring disjoint contexts for t_1 and t_2. However, for unitarity purposes, the conditional is typed additively in PUNQ (i.e., the two contexts are the same). To handle this difference, we introduce a stuck term $*$ of type $\forall X.X$ (added as a typing rule) and add the following typing rule to DLAL:

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

In what follows, when we refer to DLAL_*, we are referring to this particular extension. Notice that Corollary 6.2 remains valid for this extension as $*$ does not reduce, i.e., the normalization properties are preserved. Using the standard encoding of Booleans by the datatype $\forall X.X \multimap X \multimap X$, we associate the DLAL terms $t\ *\ t_2$ and $t\ t_1\ *$ to each conditional $\mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$, which amounts to considering the two branches separately. This encoding works well since we are only interested in bounding the derivation length of the programs in PUNQ via the derivation length of their translation into DLAL_*, and not in the precise simulation of the former system by the latter. Finally, remark that we also make use of the encoding of pairs provided in [8]. 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: $\Gamma^\star \triangleq \{x_i : (A_i)^\star \mid x_i : A_i \in \Gamma\}$.

The motivation for this encoding lies in capturing a superposition as a set of all its superposed terms. Given this set, we consider the evolution of each element, ensuring its polynomial size normalization by the DLAL system.

Lemma 6.3. *$\Gamma; \Delta \vdash \vec{t} : A$ implies $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}_*} [\vec{t}] : A^\star$.*

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

Notice also that, though not explicitly stated, the notion of depth of a typing derivation [8] is asymptotically preserved by the translation.

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

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

We will now show that the encoding preserves the size and is therefore an appropriate way to connect the complexity of the two systems.

Lemma 6.4. *For any* $\vec{t} \in$ PUNQ*, the following holds* $|[\vec{t}]| = O(|\vec{t}|)$.

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

We now show that a reduction in PUNQ corresponds to at least one reduction over its encoding in DLAL.

Lemma 6.5. *For any* $\vec{t} \in$ PUNQ*,* $\vec{t} \rightsquigarrow \vec{r}$ *implies* $[\vec{r}] \subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid [\vec{t}] \to^n \mathcal{S}_n\}$.

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

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

Theorem 6.6 (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 6.3, $\Gamma^\star; \Delta^\star \vdash_{\text{DLAL}_*} [\vec{t}] : A^\star$ holds and each term $t_i \in [t]$ has size at most $O(|t|)$. By Corollary 6.2, there is a polynomial $P \in \mathbb{N}[X]$ such that every DLAL_* term $t_i \in [\vec{t}]$ can be reduced in $N \triangleq O(P(|t_i|))$ steps. Since the size $|t_i|$ of DLAL_* terms is linearly bounded by $|\vec{t}|$, from Lemma 6.4, we have $N = O(P(|\vec{t}|))$. Therefore all the terms in $[\vec{t}]$ reduce in polynomial time on the size of $\vec{t}$. As shown in Lemma 6.5, 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 inference

In this section, we address the problem of type inference in PUNQ.

Reminder on computational complexity. The class Π_1^0 of the arithmetical hierarchy (see [36]) is defined as $\Pi_1^0 \triangleq \{\psi \mid \exists \phi \in \text{REC},\ \forall \overline{x}.(\psi(\overline{x}) \iff \exists \overline{y}.\phi(\overline{x},\overline{y}))\}$, where $\overline{x},\overline{y}$ are sequences of variables, ψ and ϕ are formulas in the language of first-order arithmetic, and REC is the class of decidable problems (recursive sets).

Let the complexity classes PTIME and 2-EXPTIME be the sets of all decision problems solvable by a deterministic Turing machine in time $O(P(n))$ and $O(2^{2^{P(n)}})$ respectively, for some polynomial $P \in \mathbb{N}[X]$, respectively. Finally, given a complexity class $\mathcal{C}$, let $\text{PTIME}^{\mathcal{C}}$ be the set of decision problems that can be computed by a polytime deterministic oracle Turing machine with oracles computing decision problems in $\mathcal{C}$.

Restriction to algebraic numbers. One problem that arises with respect to type inference concerns the representation of numbers. the multiplicative coefficients of superpositions can be any computable complex number $\alpha \in \tilde{\mathbb{C}}$. In particular, this means that checking the equality of two numbers is not computable. A natural restriction is to limit these coefficients to a (countable) subset of computable complexes where addition, multiplication, and equality are computable in polynomial time. A natural candidate is the set $\overline{\mathbb{Q}}$ of algebraic numbers, i.e., complex numbers in $\tilde{\mathbb{C}}$ that are roots of a non-zero polynomial in $\mathbb{Q}[X]$. Indeed most of the standard quantum gates can be represented as unitary matrices with algebraic coefficients. In particular, the set of Clifford+T gates, which is known to be the simplest approximatively universal fragment of quantum mechanics, lies within this restriction [1]. Hence, in what follows, we will restrict our study of type inference problems and orthogonality problems to superpositions with multiplicative coefficients in $\overline{\mathbb{Q}}$.

Linear restriction and orthogonality-free extension. Let $\text{PUNQ}_{\not\Rightarrow}$ be the type system obtained from Figure 6 by removing the rules $(\Rightarrow_i)$ and $(\Rightarrow_e)$. The set $\text{PUNQ}_{\not\Rightarrow}$ is the *linear restriction* of PUNQ, i.e., closed superpositions that can be typed without any use of the rules $(\Rightarrow_i)$ and $(\Rightarrow_e)$. In particular, it trivially holds that $\text{PUNQ}_{\not\Rightarrow} \subsetneq \text{PUNQ}$. Hence polynomial time normalization also holds for $\text{PUNQ}_{\not\Rightarrow}$ (Theorem 6.6).

Let also $\text{PUNQ}_{\not\perp}$ be the *orthogonality-free extension* of PUNQ obtained by removing all the orthogonality checks in the premises of rules $(\mathsf{if}_\sharp)$ and $(\sharp_i)$. In other words, $\text{PUNQ}_{\not\perp}$ consists of closed superpositions typable in PUNQ obtained without verifying any orthogonality conditions. It easily holds that $\text{PUNQ} \subsetneq \text{PUNQ}_{\not\perp}$ and that polynomial normalization is still valid for $\text{PUNQ}_{\not\perp}$. However, Theorem 5.6 does not hold for this system.

For a given type system $\mathcal{X} \in \{\text{PUNQ}_{\not\Rightarrow}, \text{PUNQ}, \text{PUNQ}_{\not\perp}\}$, let $\mathcal{X}(\overline{\mathbb{Q}})$ be the type system obtained by restricting $\mathcal{X}$ to superpositions whose multiplicative coefficients are all algebraic complex numbers. For example, $\text{PUNQ}(\overline{\mathbb{Q}})$ is the set of closed superpositions with algebraic coefficients that are typable using the rules of Figure 6.

Inference problems over the algebraic numbers. We now introduce type inference and orthogonality problems whose complexity will be studied in the next subsection.

Definition 7.1 (Type inference). *Given a type system $\mathcal{X}$, let $\mathsf{INF}_{\mathcal{X}}$ be the set of functions which, starting with a superposition $\vec{t}$, determines whether $\vec{t}$ is typable in $\mathcal{X}$ and outputs a concrete typing if there exists any.*

Functions in $\mathsf{INF}_{\mathcal{X}}$ can be uncomputable. Given a type $A \in \mathbb{T}_c$, we write $A \in \mathsf{INF}_{\mathcal{X}}(\vec{t})$ if there exists $F \in \mathsf{INF}_{\mathcal{X}}$ such that $F(\vec{t}) = A$. For a given complexity class $\mathcal{C}$, $\mathsf{INF}_{\mathcal{X}} \in \mathcal{C}$ holds if there exists $F \in \mathsf{INF}_{\mathcal{X}}$ such that $\{(\vec{t}, F(\vec{t})) \mid \vec{t} \in \mathcal{X}\} \in \mathcal{C}$.

Definition 7.2 (Orthogonality check). *For $\mathcal{X} \in \{\mathrm{PUNQ}_{\not\propto}, \mathrm{PUNQ}\}$, the set of orthogonal closed superpositions $\mathsf{ORTHO}_{\mathcal{X}}, \subseteq \mathsf{S}_c \times \mathsf{S}_c \times \mathbb{T}_c$ is defined by:*

$$(\vec{t}, \vec{s}, A) \in \mathsf{ORTHO}_{\mathcal{X}} \iff \vec{t} \perp_A \vec{s} \wedge A \in \mathsf{INF}_{\mathcal{X}}(\vec{t}) \wedge A \in \mathsf{INF}_{\mathcal{X}}(\vec{s})$$

Complexity of orthogonality and type inference. The following theorem shows that type inference can be decided in polynomial time in the orthogonality-free extensions of $\mathrm{PUNQ}(\overline{\mathbb{Q}})$.

Theorem 7.3. $\mathsf{INF}_{\mathrm{PUNQ}_{\not\propto}(\overline{\mathbb{Q}})} \in \mathrm{PTIME}$.

Proof. In the orthogonality-free fragment, type inference consists of a merge between Lambda-$\mathcal{S}_1$ inference and DLAL inference, plus the check for some equalities over $\overline{\mathbb{Q}}$ in rule $(\sharp_i)$ of Figure 6. Without orthogonality, Lambda-$\mathcal{S}_1$ is a simply-typed-based discipline, and it is known that type inference is PTIME-complete for the simply-typed lambda-calculus [34]. DLAL type inference can be done in PTIME [6]. This concludes the proof, as sum, product, and equalities over $\overline{\mathbb{Q}}$ can be checked in polynomial time using a suitable encoding. □

The following theorem provides the complexity of checking whether two superpositions are orthogonal.

Theorem 7.4. *The following results hold:*

1. $\mathsf{ORTHO}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})} \in \Pi_1^0$
2. $\mathsf{ORTHO}_{\mathrm{PUNQ}_{\not\propto}(\overline{\mathbb{Q}})} \in$ 2-EXPTIME

Proof. We show the two results below:

1. Consider the problem $\mathsf{ORTHO}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})}$.

$$\begin{aligned}
&(\vec{t}, \vec{s}, A) \in \mathsf{ORTHO}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})} \\
&\iff \vec{t} \perp_A \vec{s} \wedge A = \mathsf{INF}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})}(\vec{t}) \wedge A = \mathsf{INF}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})}(\vec{s}) \\
&\iff F_A(\vec{t}, \vec{s}) = 0 \wedge \forall \vec{r} \in \{\vec{s}, \vec{t}\},\ A = \mathsf{INF}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})}(\vec{r}) = \mathsf{INF}_{\mathrm{PUNQ}_{\not\propto}(\overline{\mathbb{Q}})}(\vec{r})
\end{aligned}$$

The above equivalence holds as a typable superposition can be given the same type in the orthogonality-free extension of its type system. The converse property is of course not true, as some typable superpositions of the orthogonality-free extension cannot be typed in the initial type system. By Theorem 7.3, the inference of $A = \mathsf{INF}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})_{\not\perp}}(\vec{r})$ can be done in PTIME.

Now, we show that we can encode the orthogonality condition $F_A(\vec{t},\vec{s})$ in Π_1^0 inductively on A using the definitions of Figure 4. We write $nf(\vec{t})$ for the normal form of $\vec{t}$.

$$\begin{aligned}
F_{\mathbb{B}}(\vec{t},\vec{s}) = 0 &\iff \langle nf(\vec{t})|nf(\vec{s})\rangle = 0,\\
F_{A\multimap B}(\vec{t},\vec{s}) = 0 &\iff \forall v \in \mathtt{BV}_A,\ \forall \vec{v}_1,\vec{v}_2 \in \mathtt{V}_B,\\
&\qquad nf(\vec{t}\ v) = \vec{v}_1 \wedge nf(\vec{s}\ v) = \vec{v}_2 \wedge F_B(\vec{v}_1,\vec{v}_2) = 0,\\
F_{A\times B}(\vec{t},\vec{s}) = 0 &\iff \forall \vec{v}_1,\vec{v}_2,\in \mathtt{V}_A, \forall \vec{w}_1,\vec{w}_2,\in \mathtt{V}_B,\\
&\qquad nf(\vec{t}) = \langle \vec{v}_1, w_1\rangle \wedge nf(\vec{s}) = \langle \vec{v}_2,\vec{w}_2\rangle\\
&\qquad \wedge F_A(\vec{v}_1,\vec{w}_1)F_B(\vec{v}_2,\vec{w}_2) = 0
\end{aligned}$$

All the other cases can be treated similarly. Notice that for any type A, $F_A(\vec{t},\vec{s})$ can be expressed using only universal quantifiers. Moreover, by Theorem 6.6, $nf(\vec{t})$ is in PTIME. We conclude that $\mathsf{ORTHO}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})}$ is in Π_1^0.

2. Now, let us consider the case of $\mathsf{ORTHO}_{\mathrm{PUNQ}_{\not\simeq}(\overline{\mathbb{Q}})}$. As this type system is restricted to the linear fragment of PUNQ over $\overline{\mathbb{Q}}$, any type has the structure of a vector space of finite dimension. Consequently, any formula with a universal quantifier over a value of the shape $\forall\vec{v},\ \phi(v)$ in the above proof can be reformulated equivalently into $\forall\alpha_1,\ldots,\alpha_n \in \overline{\mathbb{Q}},\ \phi(\sum_{i=1}^n \alpha_i\cdot v_i)$, provided that $b_1,\ldots,b_n$ is a basis of the considered vector space. It turns out that the decidability of orthogonality consists in checking quantifier elimination, which is known to be exponential in the number of variables (the highest dimension, hence exponential in the type) and double-exponential in the number of quantifier alternations (bounded by constant 1 in our setting), [29, Theorem 6]. Hence $\mathsf{ORTHO}_{\mathrm{PUNQ}_{\not\simeq}(\overline{\mathbb{Q}})} \in$ 2-EXPTIME. □

Combining Theorems 7.3 and 7.4, we can derive upper bounds on the complexity of type inference.

Theorem 7.5. *The following results hold:*

1. $\mathsf{INF}_{\mathrm{PUNQ}(\overline{\mathbb{Q}})} \in \mathrm{PTIME}^{\Pi_1^0}$
2. $\mathsf{INF}_{\mathrm{PUNQ}_{\not\simeq}(\overline{\mathbb{Q}})} \in \mathrm{PTIME}^{\text{2-EXPTIME}}$

Proof. Direct consequence of Theorems 7.3 and 7.4. Indeed, a possible type inference procedure is to infer types on the orthogonality-free extension and to leave the orthogonality check to an oracle. □

8 Conclusion

We have introduced PUNQ, a new quantum programming language with quantum control capabilities. PUNQ stands out as the first physically implementable language with quantum control. Supporting higher-order programs that can be superposed, the language boasts a robust type system ensuring both unitarity and polynomial-time normalization. Intriguing open questions, notably the development of a polynomial-size preserving compilation algorithm from superpositions to circuits prompt further exploration. Moreover, a potential refinement involves utilizing reducibility candidates and abstract interpretation techniques, enabling the assessment of orthogonality on an over-approximation of a function's arguments. This would allow us to alleviate the orthogonality requirements. To conclude, we claim PUNQ is a pioneering language, seamlessly combining theoretical depth with practical applicability in the realm of quantum computing.

References

1. Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Physical Review A **70**(5) (nov 2004). https://doi.org/10.1103/physreva.70.052328
2. Adleman, L.M., DeMarrais, J., Huang, M.D.A.: Quantum computability. SIAM Journal on Computing **26**(5), 1524–1540 (1997). https://doi.org/10.1137/S0097539795293639
3. Altenkirch, T., Grattage, J.: A functional quantum programming language. In: LICS 2005. pp. 249–258. IEEE Computer Society (2005). https://doi.org/10.1109/LICS.2005.1
4. Ambanis, A.: Quantum walks and their algorithmic applications. International Journal of Quantum Information **01**(04), 507–518 (2003). https://doi.org/10.1142/S0219749903000383, https://doi.org/10.1142/S0219749903000383
5. Arrighi, P., Dowek, G.: Lineal: A linear-algebraic lambda-calculus. Logical Methods in Computer Science **13**(1), 1–33 (2017). https://doi.org/10.23638/LMCS-13(1:8)2017
6. Atassi, V., Baillot, P., Terui, K.: Verification of ptime reducibility for system F terms: Type inference in dual light affine logic. Logical Methods in Computer Science **3**(4), 1–32 (2007). https://doi.org/10.2168/LMCS-3(4:10)2007
7. Badescu, C., Panangaden, P.: Quantum alternation: Prospects and problems. In: Heunen, C., Selinger, P., Vicary, J. (eds.) QPL 2015. EPTCS, vol. 195, pp. 33–42. Open Publishing Association (2015). https://doi.org/10.4204/EPTCS.195.3
8. Baillot, P., Terui, K.: Light types for polynomial time computation in lambda calculus. Information and Computation **207**(1), 41–62 (2009). https://doi.org/10.1016/j.ic.2008.08.005
9. Barber, A., Plotkin, G.: Dual intuitionistic linear logic. University of Edinburgh, Department of Computer Science, Edinburgh, UK (1996)
10. Bellantoni, S.J., Cook, S.A.: A new recursion-theoretic characterization of the polytime functions. Computational Complexity **2**, 97–110 (1992). https://doi.org/10.1007/BF01201998

11. Bennett, C.H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., Wootters, W.K.: Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. Physical Review Letters **70**(13), 1895–1899 (1993). https://doi.org/10.1103/PhysRevLett.70.1895
12. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on Computing **26**(5), 1411–1473 (1997). https://doi.org/10.1137/S0097539796300921
13. Chardonnet, K.: Towards a Curry-Howard Correspondence for Quantum Computation. Ph.D. thesis, Université Paris-Saclay (2023)
14. Chardonnet, K., de Visme, M., Valiron, B., Vilmart, R.: The many-worlds calculus. Preprint at arXiv (2022), http://arxiv.org/abs/2206.10234
15. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A.: Exponential algorithmic speedup by a quantum walk. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. ACM (jun 2003). https://doi.org/10.1145/780542.780552, https://doi.org/10.1145%2F780542.780552
16. Dal Lago, U., Masini, A., Zorzi, M.: Quantum implicit computational complexity. Theoretical Computer Science **411**(2), 377–409 (2010). https://doi.org/10.1016/j.tcs.2009.07.045
17. Díaz-Caro, A.: A quick overview on the quantum control approach to the lambda calculus. In: Ayala-Rincón, M., Bonelli, E. (eds.) Proceedings of the 16th Workshop on Logical and Semantic Frameworks with Applications (LSFA'21). Electronic Proceedings in Theoretical Computer Science, vol. 357, pp. 1–17. Open Publishing Association (2021). https://doi.org/10.4204/EPTCS.357.1
18. Díaz-Caro, A., Dowek, G., Rinaldi, J.P.: Two linearities for quantum computing in the lambda calculus. BioSystems **186**, 104012 (2019). https://doi.org/10.1016/j.biosystems.2019.104012, postproceedings of TPNC 2017
19. Díaz-Caro, A., Guillermo, M., Miquel, A., Valiron, B.: Realizability in the unitary sphere. In: LICS 2019. pp. 1–13. IEEE (2019). https://doi.org/10.1109/LICS.2019.8785834
20. Díaz-Caro, A., Malherbe, O.: Quantum control in the unitary sphere: Lambda-$\mathcal{S}_1$ and its categorical model. Logical Methods in Computer Science **18**(3), 1–32 (2022). https://doi.org/10.46298/lmcs-18(3:32)2022
21. Díaz-Caro, A., Malherbe, O.: A concrete model for a typed linear algebraic lambda calculus. To appear in Mathematical Structures in Computer Science (2023), https://arxiv.org/abs/1806.09236
22. Gaboardi, M., Marion, J., Rocca, S.R.D.: A logical account of Pspace. In: POPL 2008. pp. 121–131. ACM (2008). https://doi.org/10.1145/1328438.1328456
23. Girard, J.: Linear logic. Theoretical Computer Science **50**, 1–102 (1987). https://doi.org/10.1016/0304-3975(87)90045-4
24. Girard, J.: Light linear logic. Information and Computation **143**(2), 175–204 (1998). https://doi.org/10.1006/inco.1998.2700
25. Girard, J., Lafont, Y.: Linear logic and lazy computation. In: TAPSOFT'87. Lecture Notes in Computer Science, vol. 250, pp. 52–66. Springer (1987). https://doi.org/10.1007/BFb0014972
26. Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: a scalable quantum programming language. In: PLDI 2013. pp. 333–342. ACM (2013). https://doi.org/10.1145/2491956.2462177
27. Gurevich, Y.: Algebras of feasible functions. In: FOCS 1983. pp. 210–214. IEEE Computer Society (1983). https://doi.org/10.1109/SFCS.1983.5

28. Hainry, E., Péchoux, R., Silva, M.: A programming language characterizing quantum polynomial time. In: FoSSaCS 2023. Lecture Notes in Computer Science, vol. 13992, pp. 156–175. Springer (2023). https://doi.org/10.1007/978-3-031-30829-1_8
29. Heintz, J., Roy, M.F., Solernó, P.: Sur la complexité du principe de tarski-seidenberg. Bulletin de la Société mathématique de France **118**(1), 101–126 (1990). https://doi.org/10.24033/bsmf.2138
30. Hyland, M., de Paiva, V.: Full intuitionistic linear logic (extended abstract). Annals of Pure and Applied Logic **64**(3), 273–291 (1993). https://doi.org/10.1016/0168-0072(93)90146-5
31. Immerman, N.: Descriptive complexity. Graduate texts in computer science, Springer, New York, NY, USA (1999). https://doi.org/10.1007/978-1-4612-0539-5
32. Kempe, J.: Quantum random walks: An introductory overview. Contemporary Physics **44**(4), 307–327 (jul 2003). https://doi.org/10.1080/0010715103100011077 6, https://doi.org/10.1080%2F00107151031000110776
33. Knill, E.: Conventions for quantum pseudocode. Tech. Rep. LA-UR 96-2724, Los Alamos National Laboratory (1996), http://arxiv.org/abs/2211.02559
34. Mairson, H.G.: Linear lambda calculus and ptime-completeness. Journal of Functional Programming **14**(6), 623–633 (2004). https://doi.org/10.1017/S0956796804005131
35. Miquel, A.: A survey of classical realizability. In: TLCA 2011. Lecture Notes in Computer Science, vol. 6690, pp. 1–2. Springer (2011). https://doi.org/10.1007/978-3-642-21691-6_1
36. Odifreddi, P.: Classical recursion theory: The theory of functions and sets of natural numbers. Elsevier, Amsterdam, The Netherlands (1992)
37. Paykin, J., Rand, R., Zdancewic, S.: QWIRE: a core language for quantum circuits. In: Castagna, G., Gordon, A.D. (eds.) POPL 2017. pp. 846–858. ACM (2017). https://doi.org/10.1145/3009837.3009894
38. Péchoux, R.: Implicit computational complexity: Past and Future. Habilitation à diriger des recherches, Université de Lorraine (2020), https://hal.univ-lorraine.fr/tel-02978986
39. Procopio, L.M., Moqanaki, A., Araújo, M., Costa, F., Calafell, I.A., Dowd, E.G., Hamel, D.R., Rozema, L.A., Brukner, Č., Walther, P.: Experimental superposition of orders of quantum gates. Nature communications **6**, 7913 (2015). https://doi.org/10.1038/ncomms8913
40. Rubino, G., Rozema, L.A., Feix, A., Araújo, M., Zeuner, J.M., Procopio, L.M., Brukner, Č., Walther, P.: Experimental verification of an indefinite causal order. Science advances **3**(3), e1602589 (2017). https://doi.org/10.1126/sciadv.1602589
41. Selinger, P.: Towards a quantum programming language. Mathematical Structures in Computer Science **14**(4), 527–586 (2004). https://doi.org/10.1017/S0960129504004256
42. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Mathematical Structures in Computer Science **16**(3), 527–552 (2006). https://doi.org/10.1017/S0960129506005238
43. Van Oosten, J. (ed.): Realizability: an introduction to its categorical side. Elsevier, Amsterdam, The Netherlands (2008)
44. Voichick, F., Li, L., Rand, R., Hicks, M.: Qunity: A unified language for quantum and classical computing. Proceedings of the ACM on Programming Languages **7**(POPL), 921–951 (2023). https://doi.org/10.1145/3571225

45. Wooters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature **299**, 802–803 (1982). https://doi.org/10.1038/299802a0
46. Yamakami, T.: A schematic definition of quantum polynomial time computability. The Journal of Symbolic Logic **85**(4), 1546–1587 (2020). https://doi.org/10.1017/jsl.2020.45
47. Yao, A.C.: Quantum circuit complexity. In: FOCS 1993. pp. 352–361. IEEE Computer Society (1993). https://doi.org/10.1109/SFCS.1993.366852
48. Ying, M.: Foundations of quantum programming. Morgan Kaufmann, Burlington, MA, USA (2016)
49. Yuan, C., Villanyi, A., Carbin, M.: Quantum control machine: The limits of quantum programs as data. Preprint at arXiv (2023), 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 $\vec{t} = \mathsf{if}\ s\ \mathsf{then}\ \vec{p_1}\ \mathsf{else}\ \vec{p_2}$. Consider two cases:
  - $\Gamma_1; \Delta_1 \vdash s : \mathbb{B}$ and $\Gamma_2; \Delta_2 \vdash (\vec{p_1} \bot \vec{p_2}) : C$ 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 $\Gamma_2'; \Delta_2 \vdash (\vec{p}[u/x] \bot \vec{p}[u/x]) : C$. Notice that $\Gamma_2'; \Delta_2 \vdash \vec{p}[u/x] \bot \vec{p}[u/x]$ 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 $\Gamma_1, \Gamma_2'; \Delta_1, \Delta_2 \vdash \mathsf{if}\ s\ \mathsf{then}\ \vec{p_1}[u/x]\ \mathsf{else}\ \vec{p_2}[u/x]$ as intended.

  - Case where $\Gamma_1; \Delta_1 \vdash s : \sharp\mathbb{B}$ 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 $\vec{t} = (\mathsf{let}\ \ (x_1, x_2) = r\ \mathsf{in}\ \vec{s})$, 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 $\Gamma_1', \Gamma_2; \Delta \vdash \mathsf{let}\ \ (x_1, x_2) = r[u/x]\ \mathsf{in}\ \vec{s} : E$. Notice that in this case $(\Gamma_1', \Gamma_2) = \Gamma$ and that $\mathsf{let}\ \ (x_1, x_2) = r[u/x]\ \mathsf{in}\ \vec{s}$ is $(\mathsf{let}\ \ (x_1, x_2) = r\ \mathsf{in}\ \vec{s})[u/x]$. The case where $\Gamma_2 = (\Gamma_2', x : A)$ 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 3.5 (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 $\times_{e_\sharp}$. □

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

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,

$$\begin{aligned}[\![!(A)]\!]_\emptyset = [\![!(B)]\!]_\emptyset &=^{(\mathrm{IH})} \flat[\![B]\!]_\emptyset =^{(2)} \flat\mathsf{span}([\![B]\!]_\emptyset) =^{(3)} \flat\mathsf{span}([\![B]\!]_\emptyset) \cap \mathcal{S}_1 \\ &=^{(3)} \flat(\mathsf{span}([\![B]\!]_\emptyset) \cap \mathcal{S}_1) = \flat[\![\sharp B]\!]_\emptyset = \flat[\![A]\!]_\emptyset\end{aligned}$$

– 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,

$$\begin{aligned}[\![A[B/X]]\!]_\tau &= [\![C[B/X] \multimap D[B/X]]\!]_\tau = \{\lambda x.\vec{t} : \vec{v} \in [\![C[B/X]]\!]_\tau, \vec{t}\langle\vec{v}/x\rangle \Vdash [\![D[B/X]]\!]_\tau\} \\ &=^{(\text{IH})} \{\lambda x.\vec{t} : \vec{v} \in [\![C]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}}, \vec{t}\langle\vec{v}/x\rangle \Vdash [\![D]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}}\} \\ &= [\![C \multimap D]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}} = [\![A]\!]_{\tau\cup\{X\mapsto[\![B]\!]_\tau\}}\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

□

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 $\sigma \in [\![\Gamma,\Delta]\!]_\tau$. 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 $\sigma \in [\![\Gamma,\Delta,\{x : B\}]\!]_\tau$, we have that $\sigma = \sigma_0\langle\vec{v}/x\rangle$ for some $\sigma_0 \in [\![\Gamma,\Delta]\!]_\tau$ 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 $\sigma_0 \in [\![\Gamma,\Delta]\!]_\tau$ 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 $\sigma \in [\![\Gamma,\Delta,\{x : B, y : B\}]\!]_\tau$ 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 $\sigma \in [\![\Gamma,\Delta,\{x : B\}]\!]_\tau$, $\sigma = \sigma_0\langle\vec{v}/x\rangle$ for some $\sigma_0 \in [\![\Gamma,\Delta]\!]_\tau$ and some value $\vec{v} \in [\![B]\!]_\tau$. 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 $\langle\vec{v}/x,\vec{v}/y,\sigma_0\rangle \in [\![\Gamma,\Delta,\{x : B, y : B\}]\!]_\tau$.

(≡) 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 $\mathrm{dom}(\Delta_1,\Delta_2) \subseteq FV(\mathsf{if}\ t\ \mathsf{then}\ \vec{r}\ \mathsf{else}\ \vec{s}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1,\Gamma_2,\Delta_2)$. 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

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

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

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

  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

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

($\mathsf{if}_\sharp$) 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 $\vec{r}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ 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 $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.
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 $\mathrm{dom}(\Delta_1, \Delta_2) \subseteq FV(\mathsf{if}\ t\ \mathsf{then}\ \vec{r}\ \mathsf{else}\ \vec{s}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$ 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

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

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 $\alpha, \beta \in \tilde{\mathbb{C}}$ such that $|\alpha|^2 + |\beta|^2 = 1$. Therefore,

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

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 $(\mathrm{dom}(\Delta) \cup \{x\}) \subseteq FV(\vec{t})$ 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, $(\lambda x.\vec{t})\langle\sigma\rangle \Vdash [\![A \multimap B]\!]_\tau$.

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

- $\mathrm{dom}(\Delta_1) \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1)$ and $t\langle\sigma\rangle \Vdash [\![A \multimap B]\!]_\tau$ for all $\sigma \in [\![\Gamma_1, \Delta_1]\!]_\tau$.
- $\mathrm{dom}(\Delta_2) \subseteq FV(\vec{r}) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2)$ and $\vec{r}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ for all $\sigma \in [\![\Gamma_2, \Delta_2]\!]_\tau$.

This implies that $\mathrm{dom}(\Delta_1, \Delta_2) \subseteq FV(t\ \vec{r}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$. 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 $FV(t) \bigcap \mathrm{dom}(\sigma_2) = FV(\vec{r}) \bigcap \mathrm{dom}(\sigma_1) = \emptyset$, 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 [19, Lemma A.4].

($\Rightarrow_e$) Assuming the premises are valid, we have that

- $\mathrm{dom}(\Delta) \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma, \Delta)$ 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 $\mathrm{dom}(\Delta) \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma, \Delta) \cup \{z\}$. From the fact that $z \notin \mathrm{dom}(\Gamma, \Delta)$, 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. $\mathrm{dom}(\Delta_1) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![A]\!]_\tau$ 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 $\vec{r}\langle\sigma\rangle \Vdash [\![B]\!]_\tau$ 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:

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

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 [19, Lemmas A.10 and A.3].

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

1. $\mathrm{dom}(\Delta_1) \subseteq FV(\vec{t}) \subseteq \mathrm{dom}(\Gamma_1,\Delta_1)$ and $\vec{t}\langle\sigma\rangle \Vdash [\![\sharp(A\times B)]\!]_\tau$ 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 $\sigma \in [\![\Gamma_2,\Delta_2,\{x:\sharp A,y:\sharp B\}]\!]_\tau$.

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:

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

$$= \sum_{i=1}^{n} \alpha_i \cdot \vec{s}\langle \vec{v}_i/x, \vec{w}_i/y, \sigma_2\rangle \rightsquigarrow^* \sum_{i=1}^{n} \alpha_i \cdot \vec{z}_i \in \mathsf{span}(\llbracket C \rrbracket_\tau)$$

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 [19, Lemma D.4 and Proposition 4.1].

($\S_i$) Assuming that the premise is a valid typing judgment, we have that $FV(\vec{t}) = \mathrm{dom}(\Gamma, \Delta)$. Using the fact that $\llbracket \S B \rrbracket_\tau \triangleq \llbracket B \rrbracket_\tau$, 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 $\llbracket \Gamma, \Delta \rrbracket_\tau = \llbracket \Gamma, \S\Delta \rrbracket_\tau$ 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 $\sigma \in \llbracket \Gamma, \Delta \rrbracket$, $\vec{t_i}\langle\sigma\rangle \Vdash \llbracket A \rrbracket_\tau$. Since $\sum_{i=1}^{n} |\alpha_i|^2 = 1$, we have that

$$\Big(\sum_{i=1}^{n} \alpha_i \cdot \vec{t_i} \Big) \langle \sigma \rangle = \sum_{i=1}^{n} \alpha_i \cdot (\vec{t_i}\langle\sigma\rangle) \Vdash \llbracket \sharp A \rrbracket_\tau.$$

($\S_e$) 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 $\sigma \in \llbracket \Gamma_1, \Delta_1 \rrbracket_\tau$, $\vec{r}\langle\sigma\rangle \Vdash \llbracket B \rrbracket_\tau$.
2. $\mathrm{dom}(\Delta_2) \cup \{x\} \subseteq FV(t) \subseteq \mathrm{dom}(\Gamma_2, \Delta_2) \cup \{x\}$ and for all $\sigma \in \llbracket \Gamma_2, \Delta_2, \{x : B\} \rrbracket_\tau$, $t\langle\sigma\rangle \Vdash \llbracket A \rrbracket_\tau$.

We immediately obtain that $\mathrm{dom}(\Delta_1, \Delta_2) \subseteq FV(t[\vec{r}/x]) \subseteq \mathrm{dom}(\Gamma_1, \Delta_1, \Gamma_2, \Delta_2)$.

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

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

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

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

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

$$\llbracket \forall X.A \rrbracket_{\tau'} = \bigcap_{B \in \mathcal{S}_1} \llbracket A \rrbracket_{\tau' \cup \{X \mapsto \llbracket B \rrbracket_\emptyset\}}$$

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 $\vec{t} \in$ PUNQ and any $k, n \in \mathbb{N} - \{0\}$, if $k \geq n$, then*

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

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 $\alpha_i \in \tilde{\mathbb{C}}$ such that $\sum_i |\alpha_i|^2 = 1$ we have, by linearity,

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

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

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

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

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

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

A.3 Proofs of Section 6

Lemma 6.3. *$\Gamma; \Delta \vdash \vec{t} : A$ implies $\Gamma^\star; \Delta^\star \vdash_{\mathrm{DLAL}_*} [\vec{t}] : A^\star$.*

Proof. We proceed by induction on the structure of $\vec{t}$. Consider the following cases:

- $\vec{t} = |0\rangle$ such that $|[\vec{t}]| = 1$. By a generation lemma we have that $\Gamma = \Delta = \emptyset$ and $A = \mathbb{B}$. We have that $[\vec{t}] = [|0\rangle] = \lambda x.\lambda y.x$ for which we obtain $\vdash_{\mathrm{DLAL}_*} \lambda x.\lambda y.x : \forall\alpha.(\alpha \multimap \alpha \multimap \alpha)$. Similar for $|1\rangle$.
- If $\vec{t} = x$, by a generation lemma we either have a derivation $; x : A \vdash x : A$ for which by (Id) in DLAL_* we have that $; x : [A] \vdash_{\mathrm{DLAL}_*} x : [A]$, or $x : A; \vdash x : \S A$ by rule $\S_i$ which by $(\S_i)$ in DLAL_* we can also obtain $x : [A]; \vdash x : \S[A]$. Notice that $\S[A] = [\S A]$.
- If $t = \lambda x.\vec{s}$, then we possibly have
  - $\Gamma; \Delta, x : A_1 \vdash \vec{s} : A_2$ and by the induction hypothsis $[\Gamma]; [\Delta], x : [A_1] \vdash_{\mathrm{DLAL}_*} \vec{s_i} : [A_2]$, $\forall \vec{s_i} \in [\vec{s}]$ and therefore by rule $(\multimap_i)_{\mathrm{DLAL}_*}$ we have that $[\Gamma]; [\Delta] \vdash_{\mathrm{DLAL}_*} \lambda x.\vec{s_i} : [A_1] \multimap [A_2]$, for all i. Notice that $[A_1 \multimap A_2] = [A_1] \multimap [A_2]$.
  - $\Gamma, x_1 : A_1; \Delta \vdash \vec{s} : A_2$ can be treated similarly using rule $(\Rightarrow_i)_{\mathrm{DLAL}_*}$, where we obtain $[\Gamma]; [\Delta] \vdash_{\mathrm{DLAL}_*} [\lambda x.\vec{s}] : [A_1] \Rightarrow [A_2]$.
- If $t = s\ r$, then we either have:
  - $\Gamma_1; \Delta_1 \vdash s : A' \multimap A$ and $\Gamma_2; \Delta_2 \vdash r : A'$, in which case by the induction hypothesis there exist DLAL_* typing derivations $[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} s_i : [A' \multimap A]$, $\forall s_i \in [s]$ and $[\Gamma_2]; [\Delta_2] \vdash_{\mathrm{DLAL}_*} r_j : [A']$, $\forall r_j \in [r]$. Therefore, for each pair s_i, r_j, we have, by rule $(\multimap_e)_{\mathrm{DLAL}_*}$, a derivation $[\Gamma_1], [\Gamma_2]; [\Delta_1], [\Delta_2] \vdash_{\mathrm{DLAL}_*} s_i\ r_j : [A]$ and therefore $[\Gamma_1], [\Gamma_2]; [\Delta_1], [\Delta_2] \vdash_{\mathrm{DLAL}_*} [s\ r] : [A]$.
  - $\Gamma_1; \Delta_1 \vdash s : A' \Rightarrow A$ and $; z : C \vdash r : A'$ (or with empty context for r), which can be treated similarly, using rule $(\Rightarrow_e)_{\mathrm{DLAL}_*}$.
- $t = \mathsf{if}\ v\ \mathsf{then}\ p_0\ \mathsf{else}\ p_1$, where we consider two possibilities:

- $\Gamma_1; \Delta_1 \vdash v : \mathbb{B}$ and $\Gamma_2; \Delta_2 \vdash p_0, p_1 : A$. By induction hypothesis we have $[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} [v] : \forall\alpha.(\alpha \multimap \alpha \multimap \alpha)$ and $[\Gamma_2]; [\Delta_2] \vdash_{\mathrm{DLAL}_*} [p_0], [p_1] : [A]$. The following derivation is allowed in DLAL_*:

$$\dfrac{\dfrac{\dfrac{[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} [v] : \forall\alpha.(\alpha \multimap \alpha \multimap \alpha)}{[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} [v] : [A] \multimap [A] \multimap [A]}\,(\forall_e)_{\mathrm{DLAL}_*} \quad [\Gamma_2]; [\Delta_2] \vdash_{\mathrm{DLAL}_*} [p_0] : [A]}{[\Gamma_1, \Gamma_2]; [\Delta_1, \Gamma_2] \vdash_{\mathrm{DLAL}_*} [v\, p_0] : [A] \multimap [A]}\,(\multimap_e)_{\mathrm{DLAL}_*} \quad \vdash_{\mathrm{DLAL}_*} * : [A]}{[\Gamma_1, \Gamma_2]; [\Delta_1, \Gamma_2] \vdash_{\mathrm{DLAL}_*} [v\, p_0\, *] : [A]}\,(\multimap_e)_{\mathrm{DLAL}_*}$$

  The reasoning for the case of $[\Gamma_1, \Gamma_2]; [\Delta_1, \Delta_2] \vdash_{\mathrm{DLAL}_*} [v \ * \ p_1] : [A]$ is similar.
- $\Gamma_1; \Delta_1 \vdash \vec{v} : \sharp\mathbb{B}$ and $\Gamma_2; \Delta_2 \vdash p_0, p_1 : A$. By the induction hypothesis we have that $[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} [\vec{v}] : \forall\alpha.(\alpha \multimap \alpha \multimap \alpha)$ and $[\Gamma_2]; [\Delta_2] \vdash_{\mathrm{DLAL}_*} [p_0], [p_1] : [A]$ and therefore the previous reasoning can also be applied.

– If $t = (t_1, t_2)$, then if we applied rule $(\times_i)$, we have derivations $\Gamma_1; \Delta_1 \vdash t_1 : A_1$ and $\Gamma_2; \Delta_2 \vdash t_2 : A_2$ and $A = A_1 \times A_2$. By the induction hypothesis we have $[\Gamma_i]; [\Delta_i] \vdash_{\mathrm{DLAL}_*} [t_i] : [A_i]$, for $i \in \{1, 2\}$. By derived rule $(\otimes_i)_{\mathrm{DLAL}_*}$ we may obtain $[\Gamma_1, \Gamma_2]; [\Delta_1, \Delta_2] \vdash_{\mathrm{DLAL}_*} t_1^{(i)} \otimes t_2^{(j)} : [A_1] \otimes [A_2]$, for all $t_1^{(i)} \in [t_1]$ and all $t_2^{(j)} \in [t_2]$. This is the same as $[\Gamma_1, \Gamma_2]; [\Delta_1, \Delta_2] \vdash_{\mathrm{DLAL}_*} [\langle t_1, t_2 \rangle] : [A_1 \times A_2]$.
– In the case $\vec{t} = (\mathsf{let}\ \ (x, y) = t_1\ \mathsf{in}\ \vec{t_2})$, we consider the case of two rules:
  - For rule $(\times_e)$, we have premises $\Gamma_1; \Delta_1 \vdash t_1 : B_1 \times B_2$ and $\Gamma_2; \Delta_2, x : B_1, y : B_2 \vdash \vec{t_2} : A$. By the induction hypothesis we have the derivations $[\Gamma_1]; [\Delta_1] \vdash_{\mathrm{DLAL}_*} [t_1] : [B_1 \times B_2]$ and $[\Gamma_2]; [\Delta_2], x : [B_1], y : [B_2] \vdash_{\mathrm{DLAL}_*} [\vec{t_2}] : [A]$. Since $[B_1 \times B_2] = [B_1] \otimes [B_2]$, applying derived rule $(\otimes_e)_{\mathrm{DLAL}_*}$ we obtain that for all $u_1 \in [t_1]$, and for all $u_2 \in [\vec{t_2}]$, $[\Gamma_1, \Gamma_2]; [\Delta_1, \Delta_2] \vdash_{\mathrm{DLAL}_*}$ let u_1 be $x \otimes y$ in $u_2 : [A]$, as desired.
  - In the case of rule $(\times_{e\sharp})$, from the equalities $[\sharp(B_1 \times B_2)] = [B_1 \times B_2] = [B_1] \otimes [B_2]$ and $[\sharp A] = [A]$ we may apply precisely the same reasoning as in the above case.
– For the case $\vec{t} = \sum_i \alpha_i \cdot \vec{t_i}$, we have that for all i, we have a derivation $\vdash \vec{t_i} : A'$ such that $A = \sharp A'$, which by induction hypothesis implies $\vdash_{\mathrm{DLAL}_*} [\vec{t_i}] : [A']$. By definition, $[\vec{t}] = [\sum_i \alpha_i \cdot \vec{t_i}] = \cup_i [\alpha_i \cdot \vec{t_i}] = \cup_i [\vec{t_i}]$. Therefore we also have $\vdash_{\mathrm{DLAL}_*} : [\vec{t}] : [A]$. Notice that $[A] = [\sharp A'] = [A']$.

Using rule (W) is safe, since from $(\mathrm{Weak})_{\mathrm{DLAL}_*}$ we may perform the same operation. Contraction rule (C) is also safe, since $(\mathrm{Cntr})_{\mathrm{DLAL}_*}$ allows us to perform the exact same replacement – that of two variables in the left context. Rule $(\forall_i)$ is also safe since it exists in the exact same form in DLAL_*, with the same condition that X must not appear free in the context, and $(\forall_e)$ is also safe. The same can be argued for $(\S_i)$ and $(\S_e)$, with direct counterparts in DLAL_*, given the translation $[\S A] = \S[A]$. □

Lemma 6.4. *For any $\vec{t} \in \mathrm{PUNQ}$, the following holds $|[\vec{t}]| = O(|\vec{t}|)$.*

Proof. We consider the rules for encoding and perform a proof by induction on the structure of $\vec{t}$.

- If $\vec{t} = |0\rangle$, then $[t] = \{\lambda x.\lambda y.x\}$ and therefore $|\vec{t}| = 1$ and $|[\vec{t}]| = 3$. 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 $|t| = 1 + |\vec{s}|$ and $|[\vec{t}]| = 1 + |[\vec{s}]|$.
- If $\vec{t} = t_1\ t_2$, we have $|\vec{t}| = |t_1| + |t_2|$ and

$$\begin{aligned}|[t_1\ t_2]| &\triangleq \max\{|u_1\ u_2|, u_1 \in [t_1], u_2 \in [t_2]\}\\ &= \max\{|u_1| + |u_2|, u_1 \in [t_1], u_2 \in [t_2]\}\\ &= \sum_{i\in\{1,2\}} \max\{|u|, u \in [t_i]\} = |[t_1]| + |[t_2]|.\end{aligned}$$

  By applying the induction hypothesis we obtain the desired result.
- If $\vec{t} = \mathsf{if}\ s\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2$, we have $|\vec{t}| = 1 + |s| + \max(|p_1|, |p_2|)$, and

$$\begin{aligned}|[\vec{t}]| &\triangleq \max\big\{\max\{|u\ v_1\ *|, u \in [s], v_1 \in [p_1]\}, \max\{|u\ *\ v_2|, u \in [s], v_2 \in [p_2]\}\big\}\\ &= \max\big\{\max\{|u| + |v_1| + 1, u \in [s], v_1 \in [p_1]\}, \max\{|u| + |v_2|, u \in [s], v_2 \in [p_2]\}\big\}\\ &= \max\{1 + |u| + \max(|v_1|, |v_2|), u \in [s], v_1 \in [p_1], v_2 \in [p_2]\},\end{aligned}$$

  and so $|[t]| = O(|[s]| + |[p_1]| + |[p_2]| + 1) = O(|s| + |p_1| + |p_2| + 1) = O(|\vec{t}|)$ by applying the induction hypothesis.
- If $\vec{t} = (t_1, t_2)$, then $|\vec{t}| = |t_1| + |t_2| + 1$ and

$$\begin{aligned}|[\vec{t}]| &\triangleq \max\{|u_1 \otimes u_2|,\ u_1 \in [t_1],\ u_2 \in [t_2]\}\\ &= \max\{|u_1| + |u_2| + 1,\ u_1 \in [t_1],\ u_2 \in [t_2]\} = |[t_1]| + |[t_2]| + 1,\end{aligned}$$

  and by applying the induction hypothesis, we obtain that $|[t_1]| + |[t_2]| + 1 = O(|\langle t_1 t_2\rangle|)$.
- If $\vec{t} = \mathsf{let}\ (x, y) = t_1\ \mathsf{in}\ t_2$ then $|t| = |t_1| + |t_2| + 3$ and from the definition of $[\mathsf{let}\ (x, y) = t_1\ \mathsf{in}\ t_2]$ we have that

$$\begin{aligned}|[\vec{t}]| &\triangleq \max\{|\text{let } u_1 \text{ be } x \otimes y \text{ in } u_2|,\ u_1 \in [t_1],\ u_2 \in [t_2]\}\\ &= \max\{|u_1| + |u_2| + |x| + |y| + 1,\ u_1 \in [t_1],\ u_2 \in [t_2]\}\\ &= \max\{|u_1| + |u_2| + 3,\ u_1 \in [t_1],\ u_2 \in [t_2]\}\\ &= |[t_1]| + |[t_2]| + 3.\end{aligned}$$

- If $\vec{t} = \sum_i \alpha_i \cdot \vec{t_i}$ in canonical form, then $|\vec{t}| = \max_i |\vec{t_i}|$ and $|[\vec{t}]| = \max_i |[\alpha_i \cdot \vec{t_i}]| = \max_i |[\vec{t_i}]|$, as intended. □

Lemma 6.5. *For any* $\vec{t} \in \mathrm{PUNQ}$, $\vec{t} \rightsquigarrow \vec{r}$ *implies* $[\vec{r}] \subseteq \bigcup_{n>0}\{\mathcal{S}_n \mid [\vec{t}] \to^n \mathcal{S}_n\}$.

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

- $\vec{t} = (\lambda x.\vec{p})\ v$ and therefore $r = \vec{p}[v/x]$. Then $[\vec{t}] = \{(\lambda x.u)\ w \mid u \in [\vec{p}], w \in [v]\}$ and therefore $[\vec{t}] \to \{u[w/x] \mid u \in [\vec{p}], w \in [v]\}$. Indeed, $\vec{t} = (\lambda x.\vec{p})\ v \rightsquigarrow \vec{p}[v/x] = \vec{r}$ and therefore $[\vec{t}] \to [\vec{r}]$.

– $\vec{t} = \mathsf{if}\ v\ \mathsf{then}\ \vec{p}\ \mathsf{else}\ \vec{s} : A$. If $\Gamma; \Delta \vdash \vec{t}$ then rule (if) implies that $\Gamma; \Delta \vdash \vec{p}, \vec{s} : A$. We have $[\vec{t}] = [v\ \vec{p}\ *] \cup [v\ *\ \vec{s}] = \{u\ w * \mid u \in [v],\ w \in [\vec{p}]\} \cup \{u\ *\ w \mid u \in [v],\ w \in [\vec{s}]\}$. Considering the case $v = |0\rangle$, then $t \rightsquigarrow \vec{p}$ and $[v] = \{\lambda x.\lambda y.x\}$. We have

$$[\vec{t}] = [v\ \vec{p}\ *] \cup [v\ *\ \vec{s}] \to \{w \mid w \in [\vec{p}]\} \cup \{*\} = [\vec{p}] \cup \{*\} \supseteq [\vec{p}].$$

A similar argument can be done for $v = |1\rangle$.
If the rule applied is $(\mathsf{if}_\sharp)$ then $; \vdash \vec{v} : \sharp\mathbb{B}$ and is of the form $\vec{v} = \sum_i \alpha_i \cdot v_i$ and therefore $\vec{t} = \sum_i \alpha_i \cdot \mathsf{if}\ v_i\ \mathsf{then}\ \vec{p}\ \mathsf{else}\ \vec{s}$. We then have

$$\begin{aligned}
[t] &= \left[\sum_i \alpha_i \cdot \mathsf{if}\ v_i\ \mathsf{then}\ \vec{p}\ \mathsf{else}\ \vec{s}\right] \\
&= \bigcup_i [\mathsf{if}\ v_i\ \mathsf{then}\ \vec{p}\ \mathsf{else}\ \vec{s}] \\
&= \bigcup_i \Big([v_i\ \vec{p}\ *] \cup [v_i\ *\ \vec{s}] \Big) \\
&\to \{*\} \cup \bigcup_{i\,:\,v_i = |0\rangle} [\vec{p}] \cup \bigcup_{i\,:\,v_i = |1\rangle} [\vec{s}] =: \mathcal{S}_1.
\end{aligned}$$

In PUNQ, $\vec{t}$ reduces in one step to $\vec{r} = \sum_i \alpha_i \cdot (\delta_{v_i,|0\rangle} \cdot \vec{p} + \delta_{v_i,|1\rangle} \cdot \vec{s})$ where $\delta_{x,y}$ is the Kronecker delta between x and y. Therefore, $[\vec{r}] = \bigcup_{i\,:\,v_i = |0\rangle} [\vec{p}] \cup \bigcup_{i\,:\,v_i = |1\rangle} [\vec{s}]$ and $[\vec{r}] \subseteq \mathcal{S}_1$.

– $\vec{t} = (\mathsf{let}\ (x, y) = \langle v_1, v_2\rangle\ \mathsf{in}\ \vec{s})$, then $\vec{t} \rightsquigarrow \vec{s}[v_1/x, v_2/y]$, and we have that

$$\begin{aligned}
[\vec{t}] &= [\mathsf{let}\ (x, y) = \langle v_1, v_2\rangle\ \mathsf{in}\ \vec{s}] \\
&= \{\text{let } w \text{ be } x \otimes y \text{ in } u \mid w \in [\langle v_1, v_2\rangle],\ u_2 \in [\vec{s}]\} \\
&= \{\text{let } w_1 \otimes w_2 \text{ be } x \otimes y \text{ in } u \mid w_1 \in [v_1],\ w_2 \in [v_2],\ u \in [\vec{s}]\} \\
&\to \{u[w_1/x, w_2/y] \mid w_1 \in [v_1],\ w_2 \in [v_2],\ u \in [\vec{s}]\} \\
&= \big[\vec{s}[v_1/x, v_2/y]\big].
\end{aligned}$$

This concludes the proof. □