 Electronic Notes in Theoretical Computer Science 170 (2007) 125–138 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Quantum Patterns and Types for Entanglement and Separability

# Simon Perdrix[1](#_bookmark0)

*Leibniz Laboratory IMAG-INPG*

*Grenoble, France*

Abstract

As a first step toward a notion of quantum data structures, we introduce a typing system for reflecting entanglement and separability. This is presented in the context of classically controlled quantum computation where a classical program controls a sequence of quantum operations, *i.e.* unitary transformations and measurements acting on a quantum memory. Abstract models for such quantum computations are the Quantum Random Access Machine (*QRAM* [[5](#_bookmark7)]) and the Classically- Controlled Quantum Turing Machine (*CQTM* [[9](#_bookmark10)]). Several quantum programming languages follow this model [[1](#_bookmark3),[3](#_bookmark5),[6](#_bookmark8),[12](#_bookmark11),[13](#_bookmark12)]. Among them, the functional language defined by Valiron [[15](#_bookmark17)] is the basis for the language developed in this paper. This is work in progress.

*Keywords:* Quantum programming languages, quantum types, entanglement and separability.

# Basic Notions: Separability and Entanglement

The state of a qubit is a normalized vector in the 2-dimensional Hilbert space

C2. The state of a set of *n* qubits is generally described by a normalized vector

in the 2*n*-dimensional Hilbert space ⊗*n* C2.

*i*=1

Among all possible states of a set of qubits, some of them are separable:

Definition 1.1 For a state |*ϕ*⟩ of a set *S* of qubits, |*ϕ*⟩ is separable *iff* there exists a partition {*A, B*} of *S* (where both *A* and *B* are non empty sets) and two states |*ϕA*⟩ and |*ϕB*⟩ of the respective parts *A* and *B* such that

|*ϕ*⟩ = |*ϕA*⟩⊗ |*ϕB*⟩.

1 Email: [simon.perdrix@imag.fr](mailto:simon.perdrix@imag.fr)

1571-0661 © 2007 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2006.12.015

Definition 1.2 A quantum state |*ϕ*⟩ is entangled *iff* |*ϕ*⟩ is not separable.

For instance, a *Bell* state

1 (|01⟩− |10⟩) and the *GHZ* state

2

√

1 (|000⟩ +

2

√

|111⟩) are entangled. Entanglement is the basis for non-local operations (quan-

tum teleportation [[8](#_bookmark13)]) and constitutes a fundamental resource for measurement based quantum computation [[10](#_bookmark14),[2](#_bookmark4)]. Moreover, significant speedups (e.g. Shor algorithm [[14](#_bookmark16)]) are made possible thanks to entanglement.

Notions of entanglement and separability can be extended to qubits, and can be represented by a relation over the qubits of a set of qubits:

Definition 1.3 For a given state |*ϕ*⟩ of a set *S* of qubits, two qubits *x, y* of *S* are separable *iff* there exists a partition {*X, Y* } of the qubits of *S* such that *x* ∈ *X* and *y* ∈ *Y* , and a state |*ϕX*⟩ (|*ϕY* ⟩) of the qubits in *X* (*Y* ), such that

|*ϕ*⟩ = |*ϕX*⟩⊗ |*ϕY* ⟩.

Definition 1.4 Two qubits *x, y* are entangled *iff* they are not separable.

Definition 1.5 For a given state |*ϕ*⟩ of a set *S* of qubits, *RE*(|*ϕ*⟩) is the entanglement relation over the qubits of *S*: (*x, y*) ∈ *RE iff x* and *y* are entangled.

Lemma 1.6 *For any* |*ϕ*⟩*, RE*(|*ϕ*⟩) *is an equivalence relation.*

Proof.

* For any qubit *x*, according to the previous definitions, *x* is entangled with itself, so *RE*(|*ϕ*⟩) is *reflexive*;
* *RE*(|*ϕ*⟩) is trivially *symmetric*;
* given *x, y, z* such that (*x, y*) ∈ *RE*(|*ϕ*⟩) and (*y, z*) ∈ *RE*(|*ϕ*⟩). If *x* and *z* are separable, there exist a partition {*X, Z*} of the qubits of |*ϕ*⟩ and two states |*ϕX*⟩ *,* |*ϕZ*⟩ such that |*ϕ*⟩ = |*ϕX*⟩⊗ |*ϕZ*⟩. Since *y* is either in *X* or *Z*, either (*x, y*) ∈*/ RE*(|*ϕ*⟩) or (*y, z*) ∈*/ RE*(|*ϕ*⟩). So, by contradiction, *RE*(|*ϕ*⟩) is *transitive*.

Thus *RE*(|*ϕ*⟩) is an *equivalence relation*.

# Types for Teleportation

In order to point out the importance of handling entanglement and separa- bility in quantum programming languages, we analyze the specifications of a program for teleportation.

Teleportation can be represented as a function taking three qubits and outputting three qubits. The type of this function could be:

*teleportation* : *qbit* ⊗ *qbit* ⊗ *qbit* *qbit* ⊗ *qbit* ⊗ *qbit*

|*ϕ*⟩

*Bell*

*Bell*

*σ* |*ϕ*⟩

*qbit* × (*qbit* ⊗ *qbit*) (*qbit* ⊗ *qbit*) × *qbit*

Fig. 1. Input and output types for teleportation

where *qbit* ⊗ *qbit* is a type for any pair of two qubits, either entangled or separable.

After teleportation, the third qubit (Bob’s qubit) is not entangled with the other two because of the Bell measurement performed on them by Alice. Thus, this qubit can be manipulated independently without affecting the others. In order to represent this separability, a cartesian product, instead of a tensor product, may be used in the typing:

*teleportation* : *qbit* ⊗ *qbit* ⊗ *qbit* (*qbit* ⊗ *qbit*) × *qbit*

Furthermore, the input to this function must be separable, since teleporta- tion has no meaning if the input state is for instance a 3-qubit *GHZ* state

1 (|000⟩ + |111⟩. This constraint of input separability could be stated as

√

2

follows:

*teleportation* : *qbit* × (*qbit* ⊗ *qbit*) (*qbit* ⊗ *qbit*) × *qbit*

In order to show how such specifications could be conveyed through a typing system, we introduce a typed quantum functional language, using linear logic with two products: a tensor product for the general case and a cartesian product to represent separability constraints. *A*×*B* is then a subtype of *A*⊗*B* since separable states are also general states. Thus the cartesian product provides additional information on the state of a system, while specifying a constraint for an input to be separable.

# Terms

The language introduced here is largely inspired from Valiron’s quantum func- tional language [[15](#_bookmark17)]. A quantum program state is a triple [*Q, f, M*], where:

* + *M* is a term:

|  |  |  |
| --- | --- | --- |
| *M* | ::= | fun *P* → *M* | (*MM*) | if (*M*; *M*; *M*) | ⟨*M, M*⟩ | *U*1 *M* | *U*2 *M* |  meas *M* | let *x* = new *M* in *M* | *x* | 0 | 1 |
| *P* | ::= | *T* | ⟨*P* ⬦ *P* ⟩ |
| *T* | ::= | *x* | ⟨*T, T* ⟩ |

with *x* ranging over V a countable set of variables, and *c* ranging over

C a set of constants;

* *f* is a function from V to {1*,... , n*}, the index set of qubits in the memory;
* *Q* is a vector in a vector space C2n of dimension 2*n*.

In order to simplify the writing of the reduction rules, *pi* denotes a variable *x* such that *f* (*x*)= *i*. The quantum program state then becomes a pair [*Q, M*'] with *M*' = *M*[*pf*(*x*1 )*/x*1] *...* [*pf*(*x*n)*/xn*] if the domain of *f* is {*x*1*,... xn*}.

The syntax of terms allows: abstraction and application of functions; con-

ditional expressions and formation of pairs of terms; application of quantum operations like measurements and unitary transformations; and creation and initialization of qubits. Patterns are specified as single variables or as pairs

⟨*P*1*, P*2⟩ or ⟨*T*1*, T*2⟩. The diamond (⬦) is a syntactic way of expressing the con- straint that an argument must be separable: for instance if the teleportation function is defined as fun ⟨*x*⬦⟨*y, z*⟩⟩ → *M*, its type is *qbit*×(*qbit*⊗*qbit*) *A*, where *A* is the type of *M*.

The entanglement relation of a new qubit relates this qubit with itself. Since the information about entanglement is stated in the typing rules ([5](#_bookmark1)) by a relation over variable names, the new qubit must be immediately named. That is why the syntax always gives names to newly created qubits let *x* = new *M* in *M*.

# Reduction

Following the approach developed in [[15](#_bookmark17)] by Valiron, a call-by-value reduction system is required to tackle the consequences of no-cloning in the language: the argument must be evaluated before actually applying a function.

A value is a term *V* of the following form:

*V* ::= *x* | fun *P* → *M* | 0 | 1 | ⟨*V, V* ⟩

Due to the probabilistic nature of measurement, the reduction system is probabilistic: state →*p* state can be applied with probability *p*.

* Function application:

[*Q,* ( fun *x* → *M*) *V* ] →1 [*Q, M*[*V/x*]] If the argument is a pair:

[*Q,* ( fun *P* → *M*) *V* ] →1 [*Q, M*[*V/P* ]]

where the substitution *M*[*V/P* ] is recursively defined as: *M*[⟨*V*1*, V*2⟩*/*⟨*P*1 ⬦ *P*2⟩] = *M*[*V*1*/P*1][*V*2*/P*2] *M*[⟨*V*1*, V*2⟩*/*⟨*T*1*, T*2⟩] = *M*[*V*1*/T*1][*V*2*/T*2]

* In any case, the argument is evaluated first, before function application, then the function is evaluated:

[*Q, N*] →*p* [*Q*'*,N*']

[*Q, MN*] →*p* [*Q*'*,MN* ']

[*Q, M*] →*p* [*Q*'*,M*']

[*Q, MV* ] →*p* [*Q*'*,M*'*V* ]

* The cases for if :

[*Q,* if (0; *N*1; *N*0)] →1 [*Q, N*0]

[*Q,* if (1; *N*1; *N*0)] →1 [*Q, N*0]

[*Q, M*] →*p* [*Q*'*,M*']

[*Q,* if (*M*; *N*1; *N*0)] →*p* [*Q*'*,* if (*M* '; *N*1; *N*0)]

* Measurement of a qubit indexed by *i*:

For a given qubit *i*, *Q* is a superposition of two states, where {|0*i*⟩ *,* |1*i*⟩}

is the standard basis for qubit *i*:

*Q* = *α* |0*i*⟩⊗ *Q*0 + *β* |1*i*⟩⊗ *Q*1

Thus:

[*Q,* meas *pi*] →|*α*|2 [|0*i*⟩⊗ *Q*0*, pi*]

[*Q,* meas *pi*] →|*β*|2 [|1*i*⟩⊗ *Q*1*, pi*]

* Creation and initialisation of a qubit:

Let *i* be a fresh qubit index (*i* ∈*/ range*(*f* )):

[*Q,* let *x* = new 0 in *M* ] →1 [*Q* ⊗ |0*i*⟩ *,M*[*pi/x*]]

[*Q,* let *x* = new 1 in *M* ] →1 [*Q* ⊗ |1*i*⟩ *,M*[*pi/x*]] [*Q, N*] →*p* [*Q*'*,N*']

[*Q,* let *x* = new *N* in *M* ] →*p* [*Q*'*,* let *x* = new *N*' in *M*]

* The case for one-qubit unitary transformations: For a given qubit *i* :

[*Q, U p* ] → [*U* (*i*)*Q, p* ]

1. *i* 1 1 *i*

where *U* (*i*) is the unitary transformation which applies *U*

to qubit *i*.

1 1

* The case for two-qubit unitary transformations: Given two qubits *i, j*:

[*Q, U* ⟨*p ,p* ⟩] → [*U* (*i,j*)*Q,* ⟨*p ,p* ⟩]

1. *i j* 1 2 *i j*

# Types

There isa basic type for classical data (*bit* ) and a basic type for quantum data (*qbit* ). These types can be combined by operators of linear logic: !*A* means that *A* is duplicable; *A* *B* is the type of a function with an argument of type *A* outputting a result of type *B*. Additionally, two products of types are allowed, ⊗ and ×: the type of a pair composed of a term of type *A* and a term of type *B* is *A* ⊗ *B* or *A* × *B*. On duplicable data the two products are equivalent; on quantum data a type *A* × *B* means that the two terms of the pair are separable.

1. ::= *bit* | !*A* | *A* *A* | *A* × *A* | *B*
2. ::= *qbit* | *B* ⊗ *B*
   1. *Subtyping*

Let ≺ be an order relation over types. This ordering relation expresses the property of linear logic that a duplicable type !*A* is a subtype of the non- duplicable type *A*. Moreover subtyping is also induced by the two different products: *A* × *B* is a subtype of *A* ⊗ *B* because a separable state is a special case of a general state.

*A* ≺ *A*

*A* ≺ *B*

!*A* ≺ *B*

!*A* ≺ *B*

!*A* ≺ !*B*

*A* ≺ *A*' *B* ≺ *B*'

*A*' *B* ≺ *A* *B*'

*A* ≺ *A*' *B* ≺ *B*'

*A* ⊗ *B* ≺ *A*' ⊗ *B*'

*A* ≺ *A*' *B* ≺ *B*'

*A* × *B* ≺ *A*' × *B*'

*A* × *B* ≺ *A* ⊗ *B*

* 1. *Typing Rules*

A typing judgment is [*p*; *R*↑; *R*↓; Δ] ▶ *M* : *A*, where [*p*; *R*↑; *R*↓; Δ] is a context,

*M* a term, and *A* a type.

The context [*p*; *R*↑; *R*↓; Δ] is composed of:

* + - a function *p* which associates with each syntactic position in a term the set of variables below that position. For instance, consider the following term:

⟨⟨*a, U*1 *b*⟩*, U*2 ⟨*c, d*⟩⟩. Then *p*(*ϵ*)= {*a, b, c, d*}*, p*(1) = {*c, d*}*, p*(01) = {*a*}.

* + - an equivalence relation *R*↑ over the quantum variables of V. For the free variables of *M*, *R*↑ denotes a superset of the entanglement relation of these variables before the reduction of *M*: for any *x, y* ∈ *FV* (*M*), if (*x, y*) ∈*/ R*↑ then *x* and *y* are separable before the reduction of *M*. For bound variables, *R*↑ depends on the patterns (see abstraction rule).
    - an equivalence relation *R*↓ over the quantum variables. For the free variables of *M*, *R*↓ is an superset of the entanglement relation of these variables after the reduction of *M*. For any *x, y* ∈ *FV* (*M*), if (*x, y*) ∈*/ R*↓ then *x* and *y* are separable after the reduction of *M*.
    - and a set Δ denoted by {*x*1 : *A*1*,... , xn* : *An*} where *xi*’s are variables and

*Ai*’s are types.

The context [*p*; *R*↑; *R*↓; Δ1*,* Δ2] is a context [*p*; *R*↑; *R*↓; Δ], where Δ = Δ1 ∪ Δ2 and Δ1 ∩ Δ2 = ∅. [*p*; *R*↑; *R*↓; Δ*,x* : *A*] means [*p*; *R*↑; *R*↓; Δ*,* {*x* : *A*}]. !Δ contains only variables of duplicable types (*x* :!*A*).

In order to illustrate the respective roles of *R*↑ and *R*↓ consider the following examples:

* + - [*p*; ∅; ∅; ∅] ▶ fun ⟨*x* ⬦ *y*⟩ → ⟨*x, y*⟩ : !(*A* × *B* *A* × *B*). Here the pattern

⟨*x* ⬦ *y*⟩ impose that (*x, y*) is not in *R*↑. Since there is no free variables and no other pattern, *R*↑ = ∅ is valid.

* [*p*; ∅; {(*x, y*)}∗; *x* : *qbit, y* : *qbit*] ▶ *U*2 ⟨*x, y*⟩ : *qbit* ⊗ *qbit*, where *R*∗ means

the reflexive and transitive closure of *R*. Here, since (*x, y*) ∈*/ R*↑, the free

variables *x* and *y* are assumed to be separable, but after the reduction the unitary transformation can create entanglement between *x* and *y*, thus (*x, y*) ∈ *R*↓.

The typing rules are:

* Axiom :

If *R*↑ ⊆ *R*↓:

* Product terms :

*A* ≺ *B ax*

[*ϵ* → *x*; *R*↑; *R*↓; Δ*,x* : *A*] ▶ *x* : *B*

*FV* (*M*) is the set of free variables in term *M*. If ∀*x* ∈ *FV* (*M*)*,* ∀*y* ∈ *FV* (*N*)*,* (*x, y*) ∈*/ R*1↓ ∩ *R*2↓:

[*p*1; *R*↑; *R*1↓; Γ1*,* !Δ] ▶ *M* : *A* [*p*2; *R*↑; *R*2↓; Γ2*,* !Δ] ▶ *N* : *B*

[*p*; *R*↑; *R*1↓ ∩ *R*2↓; Γ1*,* Γ2*,* !Δ] ▶ ⟨*M, N*⟩ : *A* × *B*

×*term*

where *p*(*ϵ*)= *p*1(*ϵ*) ∪ *p*2(*ϵ*), *p*(0*.c*)= *p*1(*c*), and *p*(1*.c*)= *p*2(*c*). Otherwise:

[*p*1; *R*↑; *R*1↓; Γ1*,* !Δ] ▶ *M* : *A* [*p*2; *R*↑; *R*2↓; Γ2*,* !Δ] ▶ *N* : *B*

[*p*; *R*↑; (*R*1↓ ∩ *R*2↓); Γ1*,* Γ2*,* !Δ] ▶ ⟨*M, N*⟩ : *A* ⊗ *B*

⊗*term*

where *p*(*ϵ*)= *p*1(*ϵ*) ∪ *p*2(*ϵ*), *p*(0*.c*)= *p*1(*c*), and *p*(1*.c*)= *p*2(*c*).

Remark 5.1 According to the rules ×*term* and ⊗*term*, the relation ”*R*↓” of a pair is obtained by intersecting the relations of both elements of the pair. This construction is illustrated with the following example:

[*p*1; {(*x, y*)}∗; {(*x, y*)}∗; *x* : *qbit*] ▶ *x* : *qbit ax* [*p*2; {(*x, y*)}∗; ∅; *y* : *qbit*] ▶ meas *y* : *qbit*×!*bit*

[*p*; {(*x, y*)}∗; ∅; *x* : *qbit, y* : *qbit*] ▶ ⟨*x,* meas *y*⟩ : *qbit* × (*qbit*×!*bit*)

Even if *x* and *y* are initially entangled, the reduction of ⟨*x,* meas *y*⟩ leads to a state where *x* and *y* are separable since *y* is measured. This informa- tion of separability is in the right hand side of the tree, thus in order to transmit the information to the pair, intersection of the two relations must be done. In other words, a measurement is a ”non local” way of consum- ing entanglement: acting on only one variable may consume entanglement, whereas creation of entanglement between two variables is ”local” because it requires an operation on both variables.

* + if term:

[*p*1; *R*↑; *R*↓; Γ1*,* !Δ] ▶ *M* : *bit* [*p*2; *R*↓; *R*↓; Γ2*,* !Δ] ▶ *N*1 : *A* [*p*3; *R*↓; *R*↓; Γ2*,* !Δ] ▶ *N*0 : *A*

1 1 2 1 3

[*p*; *R*↑; (*R*↓ ∪ *R*↓)∗;Γ *,* Γ *,* !Δ] ▶ if (*M* ; *N* ; *N* ): *A if*

2 3 1 2 1 0

where *p*(*c*)= *p*2(*c*) ∪ *p*3(*c*).

Remark 5.2 Here, contrary to the case of product terms, only one of *N*1 and *N*0 will be reduced. Thus a superset of the entanglement relation of both *N*1 and *N*0 must be considered, since it is unknown which of them will actually be reduced: (*R*2↓ ∪ *R*3↓)∗ is considered.

* + Application:

[*p*1; *R*↑; *R*↓; Γ1 *,* !Δ] ▶ *N* : *A* [*p*2; *R*↑; *R*↓; Γ1*,* !Δ] ▶ fun *P* → *M* : *A* *B*

1 1 2 2

*app*

[*p*; *R*↑; *subs*(*R*↓*, p*1*,P* ); Γ1*,* Γ2 *,* !Δ] ▶ ( fun *P* → *M* )*N* : *B*

1 2

where *subs*(*R*2↓*, p*1*,P* ) uses the position function *p*1 to replace in *R*2↓

the

variables of the pattern *P* by the variables of *N*. The new position function

*p* is also obtained by replacing in *p*2 the variables of the pattern *P* according to *p*1.

Moreover, this rule is applicable only if *R*1↓ ⊆ *subs*(*R*2↑*, p*1*,P* ), i.e. entan- glement and separability of *N* are maintained in *M*.

* + Abstraction:

If *FV* (*M*) ∩ *dom*(Γ) = ∅:

(*P* : *A*) [*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ *M* : *B*

[*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ fun *P* → *M* : !(*A* *B*)

*abs*

Otherwise:

(*P* : *A*) [*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ *M* : *B*

[*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ fun *P* → *M* : *A* *B*

*abs*

where the operation introduces the variables of a pattern into the con- text, while verifying that the relation *R*↑ agrees with the structure of the pattern.

is defined as follows:

*T* [*p*; *R*↑; *R*↓; Γ*,* !Δ*,x* : *A*] ▶ *M* : *B*

*T* (*x* : *A*) [*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ *M* : *B*

*var*

where *T* is recursively defined as *T* = *ϵ* | *T* (*P* : *A*).

If ∀*x* ∈ *V ar*(*P*1)*,* ∀*y* ∈ *V ar*(*P*2)*,* (*x, y*) ∈ *R*↑:

*T* (*P*1 : *A*) (*P*2 : *B*) [*p*; *R*↑; *R*↓; Γ*,* !Δ*,* ] ▶ *M* : *C*

*T* (⟨*P ,P* ⟩ : *A* ⊗ *B*) [*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ *M* : *C*

*ent*

1 2

If ∀*x* ∈ *V ar*(*P*1)*,* ∀*y* ∈ *V ar*(*P*2)*,* (*x, y*) ∈*/ R*↑:

*T* (*P*1 : *A*) (*P*2 : *B*) [*p*; *R*↑; *R*↓; Γ*,* !Δ*,* ] ▶ *M* : *C*

*T* (⟨*P*1 ⬦ *P*2⟩ : *A* × *B*) [*p*; *R*↑; *R*↓; Γ*,* !Δ] ▶ *M* : *C*

* Measurement:

*sep*

[*p*; *R*↑; *R*↓; Δ] ▶ *M* : *qbit*

[*p*; *R*↑; *R*˜↓; Δ] ▶ meas *M* : *qbit*×!*bit*

*meas*

where *R*˜↓ = *R*↓ \ {(*x, y*) | *x* ∈ *FV* (*M*) *or y* ∈ *FV* (*M*)}.

* Unitary transformations:

[*p*; *R*↑; *R*↓; Δ] ▶ *M* : *qbit*

[*p*; *R*↑; *R*↓; Δ] ▶ *U*1

*unit1*

*M* : *qbit*

[*p*; *R*↑; *R*↓; Δ] ▶ *M* : *qbit* ⊗ *qbit*

[*p*; *R*↑; *R*˜↓; Δ] ▶ *U*2 *M* : *qbit* ⊗ *qbit*

where *R*˜↓ = (*R*↓ ∪ {(*x, y*) | *x, y* ∈ *FV* (*M*)})∗

* Initialization:

*unit2*

If *x* ∈*/ dom*(!Δ*,* Γ1*,* Γ2), and ∀*y* /= *x,* (*x, y*) ∈*/ R*2↓ and (*y, x*) ∈*/ R*2↓:

[*p*1; *R*1↑; *R*1↓; Γ1*,* !Δ] ▶ *M* : *bit* [*p*2; *R*2↑; *R*2↓; Γ2*,* !Δ*,x* : *qbit*] ▶ *N* : *A new*

[*p*2; *R*1↑; *R*2↓; Γ1*,* Γ2*,* !Δ] ▶ let *x* = new *M* in *N* : *A*

This rule is applicable only if *R*1↓ ⊆ *R*2↑.

Definition 5.3 A program [*Q, M*] is well-typed of type *A* iff there exist two equivalence relations *R*↑ and *R*↓ and a position function *p* such that [*p*; *R*↑; *R*↓; Δ] ▶ *M* : *A*, where Δ = {*x* : *qbit*|*x* ∈ *FV* (*M*)}. In this case, we write [*Q, M*]: *A*.

Example 5.4 Consider

P = fun ⟨*x* ⬦ *y*⟩ → ⟨*x, y*⟩*,*

we prove that [*Q,* P ] is well-typed and [*Q,* P ] :!(*A* × *B* *A* × *B*): Here, Δ = ∅, let *R*↑ = ∅, the description of *p* is omitted:

*ax*

[*p*; *R*↑; ∅; *x* : *A*] ▶ *x* : *A*

*ax*

[*p*; *R*↑; ∅; *y* : *B*] ▶ *y* : *B*

[*p*; *R*↑; ∅; *y* : *B, x* : *A*] ▶ ⟨*x, y*⟩ : *A* × *B* ×*term*

*var*

(*x* : *A*) [*p*; *R*↑; ∅; *y* : *B*] ▶ ⟨*x, y*⟩ : *A* × *B*

(*x* : *A*) (*y* : *B*) [*p*; *R*↑; ∅; ∅] ▶ ⟨*x, y*⟩ : *A* × *B*

(⟨*x* ⬦ *y*⟩ : *A* × *B*) [*p*; *R*↑; ∅; ∅] ▶ ⟨*x, y*⟩ : *A* × *B*

*var*

*sep*

[*p*; *R*↑; ∅; ∅] ▶ fun ⟨*x* ⬦ *y*⟩ → ⟨*x, y*⟩ : !(*A* × *B* *A* × *B*)

*abs*

Example 5.5 Consider

P = fun ⟨*x, y*⟩ → ⟨*x, y*⟩*,*

we prove that [*Q,* P ] is well-typed and [*Q,* P ] :!(*A* ⊗ *B* *A* ⊗ *B*): Here, Δ = ∅, let *R*↑ = {(*x, y*)}∗:

*ax*

[*p*; *R*↑; {(*x, y*)}∗; *x* : *A*] ▶ *x* : *A*

*ax*

[*p*; *R*↑; {(*x, y*)}∗; *y* : *B*] ▶ *y* : *B*

[*p*; *R*↑; {(*x, y*)}∗; *y* : *B, x* : *A*] ▶ ⟨*x, y*⟩ : *A* ⊗ *B* ⊗*term*

*var*

(*x* : *A*) [*p*; *R*↑; {(*x, y*)}∗; *y* : *B*] ▶ ⟨*x, y*⟩ : *A* ⊗ *B*

(*x* : *A*) (*y* : *B*) [*p*; *R*↑; {(*x, y*)}∗; ∅] ▶ ⟨*x, y*⟩ : *A* ⊗ *B*

(⟨*x, y*⟩ : *A* ⊗ *B*) [*p*; *R*↑; {(*x, y*)}∗; ∅] ▶ ⟨*x, y*⟩ : *A* ⊗ *B*

*var*

*ent*

[*p*; *R*↑; {(*x, y*)}∗; ∅] ▶ fun ⟨*x, y*⟩ → ⟨*x, y*⟩ : !(*A* ⊗ *B* *A* ⊗ *B*) *abs*

Example 5.6 Consider

P = fun ⟨*x* ⬦ ⟨*y, z*⟩⟩ → ⟨*U*2 ⟨*x, y*⟩*,* ( fun ⟨*a, b*⟩ → *a*)( meas *z*)⟩*,*

we prove that [*Q,* P ] is well-typed and [*Q,* P ] :!(*qbit* × *qbit* ⊗ *qbit* *qbit* ⊗

*qbit* × *qbit*):

Let Q = fun ⟨*a, b*⟩ → *a* be the first projection. Here, Δ = ∅, let *R*↑ = {(*y, z*)}∗:

*T*1 :

*ax*

[*p*; *R*↑; {(*y, z*)}∗; *x* : *qbit*] ▶ *x* : *qbit*

*ax*

[*p*; *R*↑; {(*y, z*)}∗; *y* : *qbit*] ▶ *y* : *qbit*

[*p*; *R*↑; {(*y, z*)}∗; *x* : *qbit, y* : *qbit*] ▶ ⟨*x, y*⟩ : *qbit* ⊗ *qbit* ⊗*term*

[*p*; *R*↑; {(*y, z*)*,* (*x, y*)}∗; *x* : *qbit, y* : *qbit*] ▶ *U*2 ⟨*x, y*⟩ : *qbit* ⊗ *qbit unit2*

*T*2 :

*ax*

[*p*; ∅; ∅; *a* : *qbit, b* :!*bit*] ▶ *a* : *qbit*

[*p*; ∅; ∅; ∅] ▶ Q : *qbit*×!*bit* *qbit abs*

*ax*

[*p*; *R*↑; {(*y, z*)}∗; *z* : *qbit*] ▶ *z* : *qbit*

*meas*

[*p*; *R*↑; ∅; *z* : *qbit*] ▶ meas *z* : *qbit*×!*bit*

*app*

[*p*; *R*↑; ∅; *z* : *qbit*] ▶ Q ( meas *z*): *qbit*

*T*1 *T*2

[*p*; *R*↑; ∅; *x* : *qbit, y* : *qbit, z* : *qbit*] ▶ ⟨*U*2 ⟨*x, y*⟩*,* Q ( meas *z*)⟩ : *qbit* ⊗ *qbit* × *qbit*

×*term*

[*p*; *R*↑; ∅; ∅] ▶ P : !(*qubit* × *qbit* ⊗ *qbit* *qbit* ⊗ *qbit* × *qbit*) *abs*

# Properties of Quantum Typing

Proofs of the following lemmas are done by structural induction:

Lemma 6.1 *If* [*p*; *R*↑*, R*↓*,* Δ] ▶ *M* : *A and A* ≺ *B, then* [*p*; *R*↑*, R*↓*,* Δ] ▶ *M* :

*B;*

Lemma 6.2 *If* [*p*; *R*↑*, R*↓*,* Δ] ▶ *M* : *A and* Γ ≺ Δ *(where* ≺ *is naturally extended), then* [*p*; *R*↑*, R*↓*,* Γ] ▶ *M* : *A;*

Lemma 6.3 *If* [*p*; *R*↑*, R*1↓*,* Δ] ▶ *M* : *A and R*1↓ ⊆ *R*2↓*, then there exists A* ≺ *B*

*such that* [*p*; *R*↑*, R*2↓*,* Δ] ▶ *M* : *B;*

Since *R*1↓ denotes the separability of the variables after the reduction, with

a weaker relation *R*2↓ (*R*1↓ ⊆ *R*2↓) the term is still typable but with a weaker

type, as it is illustrated in Lemma [6.3](#_bookmark2).

Lemma 6.4 *If* [*p*; *R*1↑*, R*↓*,* Δ] ▶ *M* : *A and R*2↑ ⊆ *R*1↑ *where R*1↑ *and R*2↑ *differs only on free variables of M, then* [*p*; *R*2↑*, R*↓*,* Δ] ▶ *M* : *A;*

For the free variables of *M*, *R*1↑ denotes the separability before the reduc- tion, thus with a stronger relation *R*2↑ (*R*2↑ ⊆ *R*1↑), *M* is still typable.

Conjecture 6.5 (Substitution) *If* [*p*; *R*↑; *R*1↓; Γ1*,* !Δ*,x* : *A*] ▶ *M* : *B and*

[*p*; *R*↑; *R*2↓; Γ2*,* !Δ] ▶ *V* : *A, then* [*p*; *R*↑; *R*1↓ ∩ *R*2↓; Γ1*,* Γ2*,* !Δ] ▶ *M*[*V/x*]: *B.*

The substitution property is not proved, as a consequence *subject reduction*

and *progress* are also conjectured but not proved.

# Toward quantum types for quantum data structures

The fact that quantum data can be entangled or separable requires additional specification that is not present in the usual notion of data structures as pro- posed in [[13](#_bookmark12),[15](#_bookmark17)]. An abstracted specification of the entanglement structure which is present or allowed within quantum data is clearly part of the infor- mation that has to be conveyed with notions of quantum data structures and quantum data types. An example given in this paper is the quantum data structure used by teleportation. This is achieved in this paper by the distinc- tion between the two kinds of patterns ⟨*P*1*, P*1⟩ and ⟨*P*1 ⬦ *P*2⟩ and between the types *A* ⊗ *B* and *A* × *B*.

A more elaborate example would be the data used during the execution of the One Way Quantum Computer [[11](#_bookmark15)]: a rectangular array of qubits, where all qubits are initially entangled with their neighbours, and where the entan- glement within this quantum data structure is consumed in a stepwise manner by successive 1-qubit measurements. An interesting special case, which would at least fit the One Way Quantum Computer, is the family of quantum data structures corresponding to graph states [[4](#_bookmark6),[7](#_bookmark9)]. This will be studied further in the scope of this work in progress, with the associated notion of types.

# Acknowledgements

I would like to thank Philippe Jorrand and Pablo Arrighi for discussions and comments.

# References

1. S. Bettelli, T. Calarco and L. Serafini, *Toward an architecture for quantum programming*, Eur. Phys. J. D, Vol. 25, No. 2, pp. 181-200 (2003)
2. V. Danos, E. Kashefi, P. Panangaden *The Measurement Calculus*, e-print

[arXiv:quant-ph/0412135](http://arxiv.org/abs/quant-ph/0412135), 2004.

1. S. J. Gay and R. Nagarajan *Communicating quantum processes*, Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jens Palsberg and Martin Abadi (Eds.), POPL 2005, January 12–14, 2005, pp. 145–157.
2. M. Hein, J. Eisert and J. H. Briegel, *Multi-party entanglement in graph states*, Phys. Rev. A 69, 062311, 2004.
3. E. Knill, *Conventions for Quantum Pseudocode*, LANL report LAUR-96-2724, 1996.
4. M. Lalire and Ph. Jorrand, *A process algebraic approach to concurrent and distributed quantum computation: operational semantics*, Proceedings of the 2nd International Workshop on Quantum Programming Languages, 2004, pp. 109–126.
5. M. Mhalla and S. Perdrix, *Complexity of Graph State Preparation*, [arXiv:quant-ph/0412071](http://arxiv.org/abs/quant-ph/0412071), 2004.
6. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
7. S. Perdrix and Ph. Jorrand, *Classically-Controlled Quantum Computation*,

[arXiv:quant-ph/0407008](http://arxiv.org/abs/quant-ph/0407008), to appear in Mathematical Structures in Computer Science.

1. S. Perdrix. *State Transfer instead of Teleportation in Measurement-based Quantum Computation*, International Journal of Quantum Information, 3(1):219-224, 2005.
2. R. Raussendorf and H. J. Briegel, *A One-Way Quantum Computer*, Phys. Rev. Lett. 86, 5188– 5191, 2001.
3. J.W. Sanders and P. Zuliani: *Quantum Programming, Mathematics of Program Construction*, Springer LNCS 1837, 80–99, 2000.
4. P. Selinger, *Towards a Quantum Programming Language*, Mathematical Structures in Computer Science, 14(4):527–586, 2004.
5. P. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal of Computing 26, pp. 1484–1509, 1997.
6. B. Valiron, *Quantum Typing*, Proceedings of the 2nd International Workshop on Quantum Programming Languages, 2004 pp. 163–178.