QPL 2005 Preliminary Version

Quantum programs with classical output streams (extended abstract)

Dominique Unruh [1]

Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe
Am Fasanengarten 5, 76131 Karlsruhe, Germany

Abstract

We show how to model the semantics of quantum programs that give classical output during their execution. That is, in our model even non-terminating programs may have output. The modelling interprets a program as a measurement process on the machines state, with the classical output as measurement result. The semantics presented here are fully abstract in the sense that two programs are equal in semantics if and only if they give the same outputs in any composition.

Key words: Quantum programming languages, denotational semantics, classical output streams.

1 Introduction

Most (quantum) algorithms take a (classical or quantum) input, calculate, and finally give a (classical or quantum) output. However, this paradigm does not capture the case where a program outputs data before its termination. Then even a non-terminating program may have outputs (possibly an infinitely long one). An example for such a program would be e.g. one that enumerates some set.

One possible way to model such a behaviour might be to model a program as a classical process operating on a quantum state, giving rise to a stochastic process (examples for this approach can be found in [3,2,8,7]). Such a language could then be augmented by operations for giving classical output, and the stochastic process would induce a probability distribution on the outputs.

However, there is a drawback to such a hybrid approach. Due to the laws of quantum mechanics, there are different probability distributions of

[1] Email: unruh@ira.uka.de

quantum states that principally cannot be distinguished. Therefore two programs might have different semantics although they have exactly the same observable behaviour. This problem was tackled by [6], where semantics of a quantum programming language where presented which did not model a classical stochastic process on quantum data, but instead represented the state of the program directly by a density operator, an established formalism describing probabilistic mixtures of quantum states. Since two density operators are equal if and only if the ensembles are indistinguishable, this yielded to fully abstract semantics in the sense that two programs have the same semantics if and only if they show the same behaviour in any larger context. However, these semantics did not have the possibility of modelling streams of classical output.

We follow the philosophy adapted by [6] and present fully abstract semantics for quantum programs with classical output streams. The idea underlying the model is to consider the execution of a program to be a physical measurement process on the state of the program. Such a measurement process takes a quantum state as input (the initial state of the system), returns a classical measurement result (the output during the programs execution) and leaves the system in a new state. Such a measurement process can be described by the established PMVM formalism (cf. Section 1.1). Of course, for a non-terminating program the notion of the state after the execution is not defined, so these programs are to be modelled by measurements without a post-measurement state, so-called POVMs (cf. Section 1.1).

We show how to combine the PMVM and POVM formalisms to allow for programs that sometimes terminate and sometimes do not. The situation is slightly complicated by the fact that non-terminating programs may have an uncountable number of possible output sequences. Fortunately, the modelling of POVMs and PMVMs presented in [1] (see also Section 1.1) is able to handle such a situation.

The most interesting construct presented here is that of loops. If non-terminating programs may have outputs, the approach of defining a loop as a least fixpoint is not straightforward. Therefore we present an alternative approach where the semantics of a loop are uniquely defined by some intuitive axioms (see Section 7).

In this extended abstract, we give complete formal definitions, but omit all proofs.

1.1 Notation and quantum mechanical formalism

Note that this summary of quantum mechanical formalism does not provide an introduction to quantum mechanics. It is mainly intended to state the nomenclature used in this paper. For a gentler introduction see e.g., [4] or [5], and [1] for the case of POVMs/PMVMs with uncountably many outcomes.

By $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{C}$ we denote the natural numbers (including 0, $\mathbb{N}_{>0}$ otherwise), the integers and the complex numbers, respectively. If A is a non-empty set, by A^* we denote the set of all finite, by A^∞ the set of all infinite, and by A^{seq} the set of all finite or infinite sequences over A. The empty word is written ε. Given two sequences a and b, ab denotes the concatenation (if a is infinite, $ab = a$).

When a countable set A is regarded as a measurable space, all its subsets are measurable. When the set A^*, A^∞, or A^{seq} is regarded as a measurable space, the set of its measurable sets is the σ-algebra generated by the sets of the form $\{\omega : \alpha \text{ is prefix of } \omega\}$ for $\alpha \in A^*$.

Pure quantum states are elements of some Hilbert space $\mathcal{H}$ with unit norm. A pure state is written $|\Psi\rangle$. Its adjoint is $\langle\Psi|$. A Hilbert space of the form $\mathbb{C}^X$ for some set X has a distinguished base, the computational base $\{|i\rangle : i \in X\}$. An operation on a pure state that results in a new pure state is represented by a unitary transformation (or in general by an isometric one, if the operation is not surjective).

To represent mixed states (i.e., states about which we have incomplete information), we use density operators, which are symmetric, positive operators on $\mathcal{H}$ of trace at most 1. A mixture of (at most countably many) states $|\Psi_i\rangle$ with probabilities p_i is represented as $\sum_i p_i |\Psi_i\rangle\langle\Psi_i|$. Each density operator corresponds to at least one mixture (with total possibility ≤ 1). Quantum processes on density operators are represented by quantum superoperators, i.e., completely positive maps on density operators which do not increase the trace. Trace-preserving superoperators we call probability preserving.

Given a density operator ρ over some composite Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$, the partial trace $\operatorname{tr}_A \rho$ is a density operator over $\mathcal{H}_B$ which represents the state of the second subsystem, if the first subsystem is lost.

Measurements on density operators are either modelled as POVMs or PMVMs. If the state of the system after the measurement is undefined, a POVM is used. A POVM $\mathcal{E}$ with outcomes in a set Ω assigns a positive symmetric operator $\mathcal{E}(A)$ on $\mathcal{H}$ to any measurable subset A of Ω, s.t. $\sum_i \mathcal{E}(A_i) = \mathcal{E}(\bigcup_i A_i)$ for any countable collection of disjoint sets (where the sum converges in the weak operator topology), $\mathcal{E}(\emptyset) = 0$, and $\operatorname{tr} \mathcal{E}(A)\rho \leq \operatorname{tr} \rho$ for all measurable $A \subseteq \Omega$ and density operators ρ. If $\operatorname{tr} \mathcal{E}(A)\rho = \operatorname{tr} \rho$ for all density operators, we call $\mathcal{E}$ probability preserving. Given a state ρ, the probability of measuring some $a \in A$ is given by $\operatorname{tr}(\mathcal{E}(A)\rho)$.

In the case that the state of the system after the measurement is defined, one has to use PMVMs. A PMVM $\mathcal{F}$ assigns a superoperator $\mathcal{F}(A)$ to each measurable $A \subseteq \Omega$, s.t. $\sum_i \mathcal{F}(A_i) = \mathcal{F}(\bigcup_i A_i)$ for any countable collection of disjoint sets (where the sum converges in the strong topology), $\mathcal{F}(\emptyset) = 0$, and $\operatorname{tr} \mathcal{F}(A)(\rho) \leq \operatorname{tr} \rho$ for all measurable $A \subseteq \Omega$ and density operators ρ. If $\operatorname{tr} \mathcal{F}(\Omega)(\rho) = \operatorname{tr} \rho$ for all density operators, we call $\mathcal{F}$ probability preserving. Given a state ρ, the probability of measuring some $a \in A$ is given by $\operatorname{tr} \mathcal{F}(A)(\rho)$, and the resulting state under that condition is $\frac{\mathcal{F}(A)(\rho)}{\operatorname{tr} \mathcal{F}(A)(\rho)}$.

2 Modelling a program's operation

We will now discuss how the operation of a program can be modelled. We first start by modelling terminating programs. Such a program takes a state (the initial state of the machine, represented by a density operator), gives some (classical) output, and returns a new density operator, the state of the machine after program execution. This behaviour can easily be modelled by a PMVM, which takes the initial to the resulting state and has a sequence of outputs as its measurement outcome. However, a non-terminating program could not be modelled thus, since such a program does not have a resulting state. Therefore, it is better modelled as a POVM, which takes the initial state and gives us a probability distribution of output sequences, but not the state after application.

We can now model terminating programs and non-terminating programs. However, we need to model programs, which do sometimes but not always terminate. Such a program we express as a mixed measurement, which we define as follows:

Definition 2.1 (Mixed measurement) Let $\mathcal{H}$ be a Hilbert space. A *mixed measurement M with outcomes in Ω over $\mathcal{H}$* is a pair $(M^{\mathrm{fin}}, M^{\mathrm{nt}})$, where M^{fin} is a PMVM over $\mathcal{H}$ and M^{nt} a POVM over $\mathcal{H}$ with outcomes in Ω.

Given a density operator ρ, the probability that the measurement terminates (i.e., there is a state after the application of the measurement), and that the outcome of the measurement lies in a measurable set $A \subseteq \Omega$, is given by $\operatorname{tr} M^{\mathrm{fin}}(A)(\rho)$, and the resulting state is $\frac{M^{\mathrm{fin}}(A)(\rho)}{\operatorname{tr} M^{\mathrm{fin}}(A)(\rho)}$.

The probability that the measurement does not terminate and has an outcome in A, is $\operatorname{tr} M^{\mathrm{nt}}(A)\rho$.

We will usually take the Hilbert space $\mathcal{H}$ as implicitly given.

Since it does not make sense to talk about measurements, where the probability of getting any result is greater than 1, we usually have to restrict mixed measurements to be probability preserving or reducing, as given by the following definition:

Definition 2.2 (Probability preserving) A mixed measurement M is *probability preserving* if for all density operators ρ it is

$$\operatorname{tr} M^{\mathrm{fin}}(A)(\rho) + \operatorname{tr} M^{\mathrm{nt}}(A)\rho \;=\; \operatorname{tr}\rho.$$

We call M *probability reducing* if for all ρ it is

$$\operatorname{tr} M^{\mathrm{fin}}(A)(\rho) + \operatorname{tr} M^{\mathrm{nt}}(A)\rho \;\le\; \operatorname{tr}\rho.$$

Using this notation, we can now model programs that may or may not terminate, by considering them to be a measurement which yields the classical output of the program as a result.

Definition 2.3 (Program) Let a countable alphabet Σ be fixed. Let ε denote the empty word in Σ^{seq}.

A *program* $\mathtt{P}$ is a probability preserving mixed measurement with values in Σ^{seq},[2] satisfying the additional requirement

$$\mathtt{P}^{\mathrm{fin}}\bigl(\{x \in \Sigma^{\mathrm{seq}} : x \text{ is infinite}\}\bigr) = 0.$$

When $\mathtt{P}$ is only probability reducing, we call $\mathtt{P}$ a *partial program* instead.

The additional requirement in this definition results from the fact that no terminating program can have an infinitely long output.

We finish this section by defining some very simple programs.

First, consider the program `noop`, which does nothing and terminates immediately. Since `noop` has a probability of 0 for non-termination on any initial state, we get $\mathtt{noop}^{\mathrm{nt}}(A) = 0$ for all A. And since the output is always ε (the empty in Σ^{seq}), we get $\mathtt{noop}^{\mathrm{fin}}(A) = 0$ for $\varepsilon \notin A$. Finally, the state is not modified, so we have $\mathtt{noop}^{\mathrm{fin}}(A) = 1$ for $\varepsilon \in A$ (since 1 is the identity on the density operators).

Second, consider the program `loop`, which does not terminate and has no output. Following a similar reasoning as with `noop`, we see that $\mathtt{loop}^{\mathrm{fin}}(A) = 0$ for all A, and $\mathtt{loop}^{\mathrm{nt}}(A) = 1$ if and only if $\varepsilon \in A$, and $\mathtt{loop}^{\mathrm{nt}}(A) = 0$ otherwise.

Finally, consider the simple program `print x` for $x \in \Sigma^*$, which outputs x and then terminates. Again, as with `noop` we have $(\mathtt{print\ x})^{\mathrm{nt}} = 0$. But, since x is output, we get $(\mathtt{print\ x})^{\mathrm{fin}}(A) = 1$ if and only if $x \in A$. This program can of course only give constant outputs; in Section 6 we show how to output the result of a measurement.

We collect these examples in the following

Definition 2.4 (Elementary programs) Let $x \in \Sigma^*$. Then the programs `noop`, `loop`, `print x` are defined by (for all measurable $A \subseteq \Sigma^{\mathrm{seq}}$)

$$\mathtt{noop}^{\mathrm{fin}}(A) = \begin{cases} 1, & \varepsilon \in A, \\ 0, & \varepsilon \notin A, \end{cases} \qquad \mathtt{noop}^{\mathrm{nt}}(A) = 0,$$

$$\mathtt{loop}^{\mathrm{fin}}(A) = 0, \qquad \mathtt{loop}^{\mathrm{nt}}(A) = \begin{cases} 1, & \varepsilon \in A, \\ 0, & \varepsilon \notin A, \end{cases}$$

$$(\mathtt{print\ x})^{\mathrm{fin}}(A) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A, \end{cases} \qquad (\mathtt{print\ x})^{\mathrm{nt}}(A) = 0.$$

It is easy to see that all these are in fact programs (as by Def. 2.3).

[2] Σ^{seq} is the set of all possible outputs of a program.

3 Elementary operations

Besides the elementary programs presented in the preceding section, a very basic kind of quantum programs is the application of unitary transformations to the state of the system. Since such an application does not terminate and does not give output, the following definition is straightforward:

Definition 3.1 (Unitary transformations on the program's state) Let U be an isometric transformation[3] on $\mathcal{H}$. Then the program $\mathtt{U}$ is defined by

$$\mathtt{U}^{\mathrm{fin}}(A)(\rho) = \begin{cases} U\rho U^\dagger, & \varepsilon \in A, \\ 0, & \varepsilon \notin A, \end{cases} \qquad \mathtt{U}^{\mathrm{nt}}(A)\rho = 0$$

for all density operators ρ over $\mathcal{H}$.

That this notion is well-defined, is shown by the following

Lemma 3.2 *Let U be an isometric transformation. Then* $\mathtt{U}$ *exists and is indeed a program.*

Most often, one does not want to apply a unitary transformation to the whole of $\mathcal{H}$, but only to some registers.

To be able to define this, we will assume for the rest of this section that $\mathcal{H}$ has the following structure:

$$\mathcal{H} = \bigotimes_{x \in V} \mathbb{C}^{T_x}.$$

Here V is a list of variable names and T_x an arbitrary countable set (the type of the variable). So $\mathcal{H}$ is decomposed into several quantum registers with names $x \in V$. Typical types might be bits, denoted by the set $\mathtt{bit} := \{0,1\}$, booleans, denoted by $\mathtt{bool} := \{\mathtt{true}, \mathtt{false}\}$, or integers, denoted by the set $\mathtt{int} := \mathbb{Z}$.

Using this decomposition, we can define the application of a unitary transformation on several variables:

Definition 3.3 (Unitary transformations on variables) Let $x_1, \dots, x_n$ be pairwise different variable names from V. Let further U be an isometric transformation on $\mathbb{C}^{T_{x_1}} \otimes \cdots \otimes \mathbb{C}^{T_{x_n}}$. Then let Φ be the canonical unitary isomorphism (that only reorders the coefficients) between $\mathcal{H}$ and

$$\mathbb{C}^{T_{x_1}} \otimes \cdots \otimes \mathbb{C}^{T_{x_n}} \otimes \bigotimes_{x \in V \setminus X} \mathbb{C}^{T_{x_n}} \qquad \text{with } X := \{x_1, \dots, x_n\}.$$

[3] Isometries are a more general case than unitary transformations. In particular, they need not be surjective. Mostly we will only use unitaries, therefore the name of the definition.

Then $U(x_1,\dots,x_n)$ is the unitary transformation $\Phi^{-1}\circ(U\otimes 1)\circ\Phi$ (here 1 is the identity on $\bigotimes_{x\in V\setminus X}\mathbb{C}^{T_{x_n}}$), and $\mathtt{U}(\mathtt{x_1},\dots,\mathtt{x_n})$ is $U(x_1,\dots,x_n)$ interpreted as a program as in Definition 3.1.

If $n=1$, we write short $\mathtt{U}\,\mathtt{x_1}$ for $\mathtt{U}(\mathtt{x_1})$.

So $\mathtt{U}(\mathtt{x_1},\dots,\mathtt{x_n})$ simply means that U is applied to the Hilbert space corresponding to the variables $x_1,\dots,x_n$.

Another very important operation is the (classical) assignment to quantum registers. E.g., when we write $\mathtt{a} := \mathtt{5}$ we mean that in the register a the value 5 is prepared. This is easily formalised using the partial trace. Consider a Hilbert space $\mathcal{H}$ decomposing into two spaces $\mathcal{H}=\mathcal{H}_A\otimes\mathcal{H}_B$. Then preparing the state $|\phi\rangle$ in $\mathcal{H}_A$ is the operation mapping a density operator ρ over $\mathcal{H}$ to $|\phi\rangle\langle\phi|\otimes\operatorname{tr}_A\rho$, where tr_A is the partial trace tracing out the space $\mathcal{H}_A$. This can easily be generalised to assignments to variables:

Definition 3.4 (Constant assignments) Let $x_1,\dots,x_n$ be pairwise different variable names from V, and $(a_1,\dots,a_n)\in\prod_{i=1}^n T_{x_i}$. Let

$$\mathcal{H}_A := \mathbb{C}^{T_{x_1}}\otimes\cdots\otimes\mathbb{C}^{T_{x_n}},$$
$$\mathcal{H}_B := \bigotimes_{x\in V\setminus X}\mathbb{C}^{T_{x_n}} \quad\text{with } X := \{x_1,\dots,x_n\},$$

and $\Phi:\mathcal{H}\to\mathcal{H}_A\otimes\mathcal{H}_B$ be as in Definition 3.3. Further tr_A denote the partial trace tracing out $\mathcal{H}_A$.

We define the superoperator S over $\mathcal{H}$ assigning $(a_1,\dots,a_n)$ to $(x_1,\dots,x_n)$:

$$S(\rho) := \Phi^\dagger\,\big(|a_1,\dots,a_n\rangle\langle a_1,\dots,a_n|\otimes\operatorname{tr}_A(\Phi\,\rho\,\Phi^\dagger)\big)\,\Phi$$

for all density operators ρ over $\mathcal{H}$.

Then $(\mathtt{x_1},\dots,\mathtt{x_n}) := (\mathtt{a_1},\dots,\mathtt{a_n})$ is the program $\mathtt{P}$ defined by

$$\mathtt{P}^{\mathrm{fin}}(E) = \begin{cases} S, & \varepsilon\in E,\\ 0, & \varepsilon\notin E,\end{cases}$$
$$\mathtt{P}^{\mathrm{nt}}(E) = 0.$$

We also write $\mathtt{x} := \mathtt{a}$ for $(\mathtt{x}) := (\mathtt{a})$.

The intuitive meaning of $\mathtt{x} := \mathtt{a}$ is to assign a to x, and the intuitive meaning of the statement $(\mathtt{x_1},\dots,\mathtt{x_n}) := (\mathtt{a_1},\dots,\mathtt{x_n})$ is to assign a_i to x_i. Note however that using this definition, only the assignment of constant values is possible. In Section 6 we show how to assign the outcome of a measurement.

Note that the constructs in this section rely on the implicit or explicit definition of the variable names V and the types T_x. To make these explicit, we may use the following convention: A program with $\mathcal{H}=\bigotimes_{x\in V}\mathbb{C}^{T_x}$, and variable names V and types T_x is prefixed by

$$\mathtt{var\ x\ :\ T_x;}$$

for each $x \in V$.

We present two examples for the constructs presented in this section:

```
var n:int; n=5;
```

This is a program over the Hilbert space $\mathcal{H} = \mathbb{C}^{\mathbb{Z}}$ which always terminates, gives no output, and where the state after the execution is $|5\rangle\langle 5|$ (independent of the initial state).

For the second example, let CNOT $:= \begin{pmatrix} 1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0 \end{pmatrix}$ operating on $\mathbb{C}^{\mathtt{bool}} \otimes \mathbb{C}^{\mathtt{bool}}$. Then

```
var a:bool; var b:bool; var c:bool;
CNOT(a,b); CNOT(c,b)
```

is a program over the Hilbert space $\mathcal{H} = \mathbb{C}^{\mathtt{bool}} \otimes \mathbb{C}^{\mathtt{bool}} \otimes \mathbb{C}^{\mathtt{bool}}$. It flips the second bit first conditioned on the first and then conditioned on the third bit. [4]

Admittedly, these constructs are quite rudimentary, they mainly serve to give a minimal set of elementary operations to be able to use the control-related constructs in the following sections. A concept of variables with a richer type-system and scoping would greatly improve the usability of the concepts described here and would be interesting future work.

Note further that it seems very restrictive that only constant assignments are possible in this language. However, in Section 6 it is shown how to assign the outcomes of measurements to variables.

4 Probabilistic sum

The simplest operation on programs is the probabilistic sum. Let `P` and `Q` be programs, and $p \in [0,1]$, then $\mathtt{P} \oplus_p \mathtt{Q}$ denotes the program resulting by running `P` with probability $(1-p)$ and `Q` with probability p. It is easy to see that this intuition is satisfied by the following definition:

Definition 4.1 (Binary probabilistic sum) Let `P` and `Q` be programs (or partial programs), and $p \in [0,1]$. Then we define the program (the partial program) $\mathtt{P} \oplus_p \mathtt{Q}$ by

$$(\mathtt{P} \oplus_p \mathtt{Q})^{\mathrm{fin}} := (1-p)\,\mathtt{P}^{\mathrm{fin}} + p\,\mathtt{Q}^{\mathrm{fin}},$$
$$(\mathtt{P} \oplus_p \mathtt{Q})^{\mathrm{nt}} := (1-p)\,\mathtt{P}^{\mathrm{nt}} + p\,\mathtt{Q}^{\mathrm{nt}}.$$

We can even easily generalise this definition to an arbitrary number of summands:

[4] In this example, we use the sequential composition of `CNOT(a,b)` and `CNOT(c,b)`. This we be formally introduced only in Section 5.

Definition 4.2 (Probabilistic sum) Let I be a countable set. Let $\mathtt{P}_i$ $(i \in I)$ be programs (partial programs), and $p_i \in [0,1]$ $(i \in I)$ with $\sum_i p_i = 1$. Then the *probabilistic sum* is the program (partial program) $\bigoplus_i p_i \mathtt{P}_i$ defined by

$$\Bigl(\bigoplus_i p_i \mathtt{P}_i\Bigr)^{\mathrm{fin}} := \sum_i p_i \mathtt{P}_i^{\mathrm{fin}},$$
$$\Bigl(\bigoplus_i p_i \mathtt{P}_i\Bigr)^{\mathrm{nt}} := \sum_i p_i \mathtt{P}_i^{\mathrm{nt}}.$$

A question naturally arising would be, whether the probabilistic sum is always defined, especially for infinitely many summands. The following lemma answers this question positively.

Lemma 4.3 *Let I be a countable set. If all $\mathtt{P}_i$ $(i \in I)$ are programs, and $\sum_{i\in I} p_i = 1$, then $\bigoplus_i p_i \mathtt{P}_i$ exists, is uniquely defined and a program.*

If all $\mathtt{P}_i$ are partial programs, and $\sum_{i\in I} p_i \le 1$, then $\bigoplus_i p_i \mathtt{P}_i$ exists, is uniquely defined and a partial program.

Example: Using this constructor, and the program `print` from the preceding section, we can formulate a program, which outputs a random bit:

$$\mathtt{print\ 0} \oplus_{\frac{1}{2}} \mathtt{print\ 1}.$$

5 Sequential composition

Probably the most important construction in any imperative programming language is sequential composition, i.e., the successive application of programs. To achieve this goal, we first formulate the composition of mixed measurements.

Let P and Q be mixed measurements. What does the composition QP (Q applied after P) mean intuitively? First we measure P, yielding outcome x_P. Then, if P terminates, we measure Q, yielding outcome x_Q. The overall outcome of this experiment shall then be (x_P, x_Q) or x_P (depending on whether Q has been applied or not). This intuition easily gives us the following properties of QP, which turn out to define QP (cf. Definition 2.1):

Definition 5.1 (Sequential composition of mixed measurements) Let P and Q be mixed measurements with outcomes in Ω_P resp. Ω_Q. Then QP is the mixed measurement with outcomes in $(\Omega_P \times \Omega_Q) \cup \Omega_P$ satisfying the following equalities for all density operators ρ and all measurable $A_P \subseteq \Omega_P$, $A_Q \subseteq \Omega_Q$:

$$(QP)^{\mathrm{fin}}(A_P \times A_Q)(\rho) = Q^{\mathrm{fin}}(A_Q)\bigl(P^{\mathrm{fin}}(A_P)(\rho)\bigr),$$
$$\operatorname{tr}(QP)^{\mathrm{nt}}(A_P \times A_Q)\rho = \operatorname{tr} Q^{\mathrm{nt}}(A_Q)P^{\mathrm{fin}}(A_P)(\rho),$$
$$\operatorname{tr}(QP)^{\mathrm{nt}}(A_P)\rho = \operatorname{tr} P^{\mathrm{nt}}(A_P)\rho.$$

The following lemma justifies calling these properties a definition:

Lemma 5.2 *Let P and Q be mixed measurements with outcomes in Ω_P resp. Ω_Q.*

(i) *If QP exists, it is uniquely defined by Definition 5.1.*

(ii) *If Ω_P and Ω_Q can be embedded in compact metrisable spaces, the composition QP exists.*

(iii) *If $\Omega_P = \Omega_Q = \Omega_\Sigma$, the composition QP exists.*

(iv) *If P and Q are probability preserving (reducing), so is QP (if existent).*

At a first glance, one might think that this definition already gives us the sequential composition of programs. However consider the following example: Let $\mathtt{P}_s$ output $s \in \Sigma^*$. Then we expect the composition of $\mathtt{P}_{\mathtt{ab}}$ and $\mathtt{P}_{\mathtt{c}}$ to have the same operational semantics as the composition of $\mathtt{P}_{\mathtt{a}}$ and $\mathtt{P}_{\mathtt{bc}}$, namely $\mathtt{P}_{\mathtt{abc}}$. However, using Definition 5.1 we get $\mathtt{P}_{\mathtt{c}}\mathtt{P}_{\mathtt{ab}}$, which yields with probability 1 the outcome $(\mathtt{ab}, \mathtt{c}) \neq \mathtt{abc}$. Similarly, $\mathtt{P}_{\mathtt{bc}}\mathtt{P}_{\mathtt{a}}$ outputs $(\mathtt{a}, \mathtt{bc}) \neq \mathtt{abc}$. This problem can be solved by defining the composed program $\mathtt{P}_{\mathtt{a}}; \mathtt{P}_{\mathtt{bc}}$ to result from applying the composed mixed measurement $\mathtt{P}_{\mathtt{c}}\mathtt{P}_{\mathtt{ab}}$ and then "forget" about the structure of the outcome, i.e., we map $(\mathtt{ab}, \mathtt{c})$ to $\mathtt{abc}$, and more generally (x, y) to the concatenation xy.

In order to formalise this idea, we first have to define what it means to apply a function f to the result of a mixed measurement P. Note that $P^{\text{fin}}(A)$, $P^{\text{nt}}(A)$ describe the behaviour of the measurement restricted to the case that the outcome x lies in A. Then $P^{\text{fin}}(f^{-1}(A))$, $P^{\text{nt}}(f^{-1}(A))$ describe the behaviour of the measurement restricted to the case that $f(x) \in A$. Considering this, one easily understands the following definition:

Definition 5.3 (Function application to mixed measurements) Let P be a mixed measurement with outcomes in Ω. Let further $f : \Omega \to \tilde{\Omega}$ be a measurable function. Then we define the mixed measurement $f(P)$ with outcomes in $\tilde{\Omega}$ by setting (for all measurable $A \subseteq \tilde{\Omega}$):

$$\big(f(P)\big)^{\text{fin}}(A) := P^{\text{fin}}\big(f^{-1}(A)\big),$$
$$\big(f(P)\big)^{\text{nt}}(A) := P^{\text{nt}}\big(f^{-1}(A)\big).$$

If f has a domain containing but not equaling Ω, we slightly generalise the definition by setting $f(P) := f|_\Omega(P)$.

The following lemma is then obvious:

Lemma 5.4 *Let P be a mixed measurement with outcomes in Ω. Let further $f : \Omega \to \tilde{\Omega}$ be a measurable function.*

(i) *The mixed measurement $f(P)$ exists and is uniquely defined by Definition 5.3.*

(ii) *If P is probability preserving (reducing), so is $f(P)$.*

We now have the means to formulate the

Definition 5.5 (Sequential composition of programs) Let flatten be the function taking a (finite or infinite) sequence over Σ^{seq} and returning the concatenation of the elements of the sequence.[5]

Then we define the sequential composition $\mathtt{P};\mathtt{Q}$ of two programs (partial programs) by

$$\mathtt{P;Q} := \mathrm{flatten}(\mathtt{QP}),$$

where on the right hand side $\mathtt{P}$ and $\mathtt{Q}$ are treated as mixed measurements.

We are now able to formulate simple programs like

`print a; print b`	(outputs ab),
`print a; loop`	(outputs a, but does not terminate),

However, two questions arise naturally: Is $\mathtt{P};\mathtt{Q}$ in fact a program, and is the operator `;` associative, as one would expect? The following lemma answers these questions positively and thus justifies Definition 5.5.

Lemma 5.6 *Let* $\mathtt{P}$, $\mathtt{Q}$, $\mathtt{R}$ *be programs (partial programs). Then*

(i) $\mathtt{P;Q}$ *exists, is uniquely defined, and is a program (partial program).*

(ii) *It is* $\{\mathtt{P;Q}\}\mathtt{;R} = \mathtt{P;}\{\mathtt{Q;R}\}$.[6]

(iii) *It is* $\mathtt{P;noop} = \mathtt{noop;P} = \mathtt{P}$.

Using the notion of composition, we can now formalise the claim that the semantics presented here are fully abstract:[7]

Lemma 5.7 *Let* $\mathtt{P} \neq \mathtt{Q}$ *be programs. Then there are programs* $\mathtt{S},\mathtt{T}$ *and a measurable set* $A \subseteq \Sigma^{\mathrm{seq}}$ *of outputs, s.t. the probabilities that* $\mathtt{S;P;T}$ *or* $\mathtt{S;Q;T}$ *has an output in* A *are different for any initial state* ρ*. Formally:*

$$\begin{aligned}&\mathrm{tr}(\mathtt{S;P;T})^{\mathrm{fin}}(A)(\rho) + \mathrm{tr}(\mathtt{S;P;T})^{\mathrm{nt}}(A)\rho\\ \neq\ &\mathrm{tr}(\mathtt{S;Q;T})^{\mathrm{fin}}(A)(\rho) + \mathrm{tr}(\mathtt{S;Q;T})^{\mathrm{nt}}(A)\rho.\end{aligned}$$

6 Branching programs

It would be quite hard to formulate interesting program code without the possibility of branching. We will first discuss the simplest constructor for

[5] In this definition, we use flatten only on $(\Sigma^{\mathrm{seq}})^2$. However, in Definition 7.1 we will need to apply flatten to $(\Sigma^{\mathrm{seq}})^{\mathrm{seq}}$.

[6] Note that for grouping programs, we use curly braces instead of parentheses, as common in many programming languages like C, Java, etc.

[7] Formally, to state that our semantics are fully abstract, one would need operational semantics to compare with. However, the lemma shows that with respect to a reasonable notion of observable behaviour the semantics are indeed fully abstract.

branching programs, the `if`-statement, and then proceed to a more general construction, the `switch`-statement.

Let B be a PMVM with two possible outcomes: `true` and `false`, representing a Boolean test on the state of the program (e.g., measuring two registers, and returning whether they are equal). Then the program "`if (B) P else Q`" has the following intuitive representation: First, we apply the measurement `B`, then, if the outcome is `true`, we run `P`, otherwise `Q`. The output of "`if (B) P else Q`" is that of `P` or `Q`, respectively.

Using the semantics described in Definition 2.1, we easily see that this behaviour is captured by the following

Definition 6.1 (Branching using `if`) Let `B` be a probability preserving (reducing) PMVM with outcomes in $\{\texttt{true},\texttt{false}\}$. Let further `P` and `Q` be programs (partial programs). Then

$$\texttt{R} := \texttt{if (B) P else Q}$$

is the program (partial program) satisfying (for all measurable $A \subseteq \Sigma^{\text{seq}}$ and all density operators ρ):

$$\begin{aligned}\texttt{R}^{\text{fin}}(A)(\rho) &= \texttt{P}^{\text{fin}}(A)\big(\texttt{B}(\texttt{true})(\rho)\big) + \texttt{Q}^{\text{fin}}(A)\big(\texttt{B}(\texttt{false})(\rho)\big)\\ \operatorname{tr}\texttt{R}^{\text{nt}}(A)\rho &= \operatorname{tr}\texttt{P}^{\text{nt}}(A)\texttt{B}(\texttt{true})(\rho) + \operatorname{tr}\texttt{Q}^{\text{nt}}(A)\texttt{B}(\texttt{false})(\rho)\end{aligned}$$

Further "`if (B) P`" stands for "`if (B) P else noop`".

This definition is supported by the following

Lemma 6.2 *If `P` and `Q` are programs (partial programs), and `B` a probability preserving (reducing) PMVM with outcomes in $\{\texttt{true},\texttt{false}\}$, then "`if (B) P else Q`" and "`if (B) P`" exist, are uniquely defined, and are programs (partial programs).*

This lemma follows easily from the more general Lemma 6.4.

As an example, we formulate a small program which sets a qubit to 0 and outputs its previously measured content, using only measurements and unitary operations:

```
var a:bit; if (a=0) print 0 else { NOT a; print 1 }
```

Here `NOT` denotes the bit-flip, and `a=0` is the PMVM measuring `a` and yielding `true` if and only if the outcome is 0. The formal notation for elementary tests like `a=0` is introduced in Section 6.1.

A more general construct which has `if` as a special case is the `switch`-statement. Later in this section we will see that its additional power is helpful in formulating quantum programs.

Consider a PMVM `M` with outcomes in a countable set C, and a family of programs `P(c)` indexed by $\texttt{c} \in C$. Then we can interpret the program `switch`

`(M as c) P(c)` to describe the following experiment: First, we measure the program's state using `M`. Let $\mathtt{c} \in C$ denote the outcome of that measurement. Then we execute `P(c)`. Quite analogous to Definition 6.1, we get

Definition 6.3 (Branching using `switch`) Let `M` be a probability preserving (reducing) PMVM with outcomes in a countable set C. Let further each `P(c)` ($\mathtt{c} \in C$) be a program (partial program). Then the program (partial program) $\mathtt{R} :=$ `switch (M as c) P(c)` is defined by satisfying for all measurable $A \subseteq \Sigma^{\mathrm{seq}}$ and all density operators ρ:

$$\mathtt{R}^{\mathrm{fin}}(A)(\rho) = \sum_{\mathtt{c}\in C} \bigl(\mathtt{P(c)}\bigr)^{\mathrm{fin}} \bigl(\mathtt{M}(\{c\})(\rho)\bigr)$$

$$\operatorname{tr} \mathtt{R}^{\mathrm{nt}}(A)\rho = \operatorname{tr} \sum_{\mathtt{c}\in C} \bigl(\mathtt{P(c)}\bigr)^{\mathrm{nt}} \mathtt{M}(\{c\})(\rho)$$

The convergence notion used in these equations is that of weak convergence.[8]

Lemma 6.4 *Under the conditions of Definition 6.3, the following holds:*

(i) *If all `P(c)` are programs (partial programs), and `B` is probability preserving (reducing), then "`switch (M as c) P(c)`" is a program (partial program).*

(ii) *If `B` has outcomes in $\{\mathtt{true}, \mathtt{false}\}$, then:*

```
switch (M as b) P(b)   =   if (M) P(true) else P(false)
```

The reader may now ask, what we need such a `switch`-statement for, since a finite branching can be realised using `if`-statements, and an infinite branching does not really reflect the possibilities in practical programming. However, the following example may show the use of the `switch`-statement as a tool in specifying program behaviour.

In Definition 2.4 we have introduced the program `print x`, which outputs the constant word `x`. In many cases this is not sufficient, since one may want to simply output the result of a measurement. This can easily be formulated using only `switch` and `print`. Assume `M` to be a PMVM with outcomes in A, and $f : A \to \Sigma^*$ to assign labels to the outcomes. Then the following program measures `M` and outputs the outcome:

```
switch (M as x) print f(x)
```

Similarly, the assignment of constant values from Definition 3.4 can be extended to allow for assignments of measurement outcomes:

```
switch (M as x) a := x
```

[8] Since the sums are increasing norm-bounded sequences of symmetric operators, strong, weak and ultra-weak convergence coincide.

assigns the outcome of measuring `M` to variable `a`.

To ease reading of the program code, we will often write the shorter `P(M)` instead of the less handy `switch (M as c) P(c)`. So the programs just presented will get the more intuitive forms `print f(M)` and `a := M`.

Note however that when using this shorthand notation, one has to ensure that no ambiguity ensues. E.g., one must keep in mind that "`P(M); Q(M)`" shall always denote "`switch (M as c) P(c); switch (M as c) Q(c)`", and that a statement containing two implicit `switch`-statements is only well-defined if the PMVMs commute. So `P(M,N)` could be

```
switch (M as m) switch (N as n) P(m,n)
```

or

```
switch (N as n) switch (M as m) P(m,n)
```

which are only identical, if M and N commute. So if in doubt, explicitly write `switch`. Also, a program like `a:=f(b)` could be read as `switch (f(b) as x) a:=x` (measure `f(b)` and assign the outcome to `a`) or `switch (b as x) a:=x` (measure `b` and assign `f`(b) to `a`). Our convention is to assume the latter case.

6.1 Elementary tests

In order to be able to use the above `if` statement, we still need some means to specify elementary tests. In the preceding section, we just assumed some PMVM with boolean outcomes to be given, here we will present a method how to specify these. Similarly to the case of unitary transformations, we can define measurements on functions of variables:

Given some variables $x_1, \dots, x_n$ and a function f on the types of these variables, we define $f(x_1, \dots, x_n)$ to be the measurement that measures the value of $f(x_1, \dots, x_n)$ without measuring $x_1, \dots, x_n$ (e.g., measuring $x_1 \oplus \cdots \oplus x_n$ would measure the parity of $x_1, \dots, x_n$ without performing a complete measurement). With such a measurement, getting measurement result m means projecting the state onto the subspace $\mathcal{H}_m$ of states where $f(x_1, \dots, x_n) = m$. We mold this into a formal definition:

Definition 6.5 (Elementary measurements on variables) Let $x_1, \dots, x_n$ be pairwise different variable names from V, and M a countable set. Further let $f : T_{x_1} \times \cdots \times T_{x_n} \to V$ be a function.

Then for $m \in M$, let B_m be the set of all elements e of the computational basis of $\mathcal{H} = \otimes_{x \in V} \mathbb{C}^{T_x}$ satisfying:

$$e = \otimes_{x \in V} |v_x\rangle \qquad \text{with} \qquad f(v_{x_1}, \dots, v_{x_n}) = m.$$

Then we can define $\mathcal{H}_m$ to be the subspace of $\mathcal{H}$ generated by B_m, and P_m to be the orthogonal projection onto $\mathcal{H}_m$. And finally this defines a PMVM $\mathtt{f}(\mathtt{x_1}, \dots, \mathtt{x_n})$:

$$\mathtt{f}(\mathtt{x_1}, \dots, \mathtt{x_n})(A)(\rho) := \sum_{m \in A} P_m \rho P_m^\dagger$$

for any density operator $\rho \in \mathcal{H}$ and any $A \subseteq M$.

The following lemma states the well-definedness of the above construct:

Lemma 6.6 *With the notation of Definition 6.5, if the variables $x_1, \dots, x_n$ are pairwise different, $\mathtt{f}(\mathtt{x_1}, \dots, \mathtt{x_n})$ is a probability preserving PMVM with outcomes in M.*

With $M = \{\mathtt{true}, \mathtt{false}\}$, the above construct is suitable for use with the `if` statement. We give an example:

```
var a:bit; var b:bit;
a:=0; b:=0;
H₂a; H₂b;
if (a=b) NOT a;
```

Here H_2 denotes the Hadamard-transform on one qubit, and `NOT` the bit-flip. If the test `a=b` fails, `a` and `b` are entangled to have opposite values. Otherwise, they are entangled to have the same value, but `a` gets flipped, so after the `if` statement they have opposite values, too. So the above example generates an EPR pair.

Of course, such PMVMs can also be used in conjunction with switch. So e.g.,

```
var i:int; switch (i as c) { case (i*i=4):  print "X"; }
```

and

```
var i:int; switch (i*i as c) { case (i=4):  print "X"; }
```

are different programs. While the first completely measures `i`, the second directly measures the square of `i`, i.e., ignores the sign, so if e.g., `i` is in superposition between 2 and -2, this superposition is not destroyed.

Like in Section 3, the notation for elementary measurements given here is only rudimentary. The development of more powerful concepts would be interesting future work.

7 Loops

In this exposition, one control structure is still missing, without which hardly any useful program can be written: the loop.

Assume that a program `P` and a probability preserving PMVM `B` with outcomes in $\{\mathtt{true}, \mathtt{false}\}$ are given (cf. Definition 6.1). Then the program `while (B) P` shall intuitively represent the following experiment: Repeatedly measure `B`. While `B` yields `true`, apply `P`. When `B` yields `false`, stop. The overall output shall be the concatenation of the outcomes of all invocations of `P`.

In order to get a formal definition of the above `while`-program, let us first consider the intermediary case, where the outcome of the loop is not the concatenation of the outputs of `P`, but the possibly infinite list of these outputs. I.e., let `R` denote the following experiment: Repeatedly measure `B`. While `B` yields `true`, apply `P`. When `B` yields `false`, stop. The overall output shall be the (finite or infinite) sequence of the outcomes of all invocations of `P`.

It turns out to be quite difficult to write the infinite process in the intuitive definition of `R` in terms of sums and products of operators (as we did e.g. in Definitions 5.1 and 6.3), since no intuitive notion of convergence springs to mind where the following would be meaningful:[9]

$$\underbrace{\texttt{if (B) \{P; if (B) \{P; }\cdots\texttt{ if (B) \{P;}}_{n\text{ times}}\texttt{ loop\}}\cdots\texttt{\}\}} = \texttt{while (B) P}$$

Another common approach would be to define `R` to be the lowest fixpoint of $X \mapsto$ `if (B) { P;` X `}`. However, at fails when using the natural operator-theoretic ordering on mixed measurements.[10] Of course, there may be other orders which can be used for defining loops of programs with classical output.

We will try an alternative approach and postulate some axioms, which should hold for `R`, and will then show that these indeed define `R`.

First, observe that always one (and only one) of the following cases is bound to occur:

(i) The loop terminates after $n \geq 0$ invocations of `P`.

(ii) The n-th invocation of `P` does not terminate ($n \geq 1$).

(iii) Every invocation of `P` terminates, but `B` always yields `true` (so the loop does not terminate either).

Note that the only case where `R` terminates is the first one. Therefore we can at least write down, what we expect from $\mathtt{R}^{\mathrm{fin}}$: For any $n \geq 0$, any initial state ρ, and all measurable $A_i \subseteq \Sigma^{\mathrm{seq}}$, it holds

$$\mathtt{R}^{\mathrm{fin}}(A_1 \times \cdots \times A_n)(\rho) = \mathtt{B}(\mathtt{false}) \circ \prod_{i=1}^{n} \left(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\right)(\rho).$$

Here $\prod_{i=1}^{n} X_i$ means the composition $X_n \circ \cdots \circ X_1$.

[9] In fact, there are metrics on the set of partial programs such that the above statement is meaningful and equivalent to Definition 7.1, and even allows the definition of `while (B) P` where `B` and `P` are only probability reducing. However, we believe that these metrics can not as easily be justified as the axiomatic approach below, and therefore present the (probably more natural) axiomatic approach instead.

[10] The natural operator-theoretic order is given by: $A \geq B$ if $A - B$ is a mixed measurement (all mixed measurements are positive). The problem consist in having a difference between the least mixed measurement 0 and the program `loop`. Both are solutions to the equation $X =$ `if (B) { P;` X `}`, but `loop` is the fixpoint we are looking for, while 0 is the least one.

Similarly, the case where $\mathtt{R}$ does not terminate, but has only a finite number of outputs is covered by case ii, which we can formalise as follows:

$$\operatorname{tr} \mathtt{R}^{\mathrm{nt}}(A_1 \times \cdots \times A_n)\, \rho \\ = \begin{cases} \operatorname{tr}\ \mathtt{P}^{\mathrm{nt}}(A_n)\ \mathtt{B}(\mathtt{true}) \circ \prod_{i=1}^{n-1} \big(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\big)\,(\rho), & n \geq 1 \\ 0, & n = 0. \end{cases}$$

The last case is more difficult. In order to approach that case, we first note that by requiring $\mathtt{R}$ to be probability preserving (which seems a sensible thing to do, since both $\mathtt{P}$ and $\mathtt{B}$ are), we get

$$\operatorname{tr} \mathtt{R}^{\mathrm{nt}}\big((\Sigma^{\mathrm{seq}})^{\infty}\big)\, \rho = 1 - \operatorname{tr} \mathtt{R}^{\mathrm{fin}}\big((\Sigma^{\mathrm{seq}})^{*}\big)(\rho) - \operatorname{tr} \mathtt{R}^{\mathrm{nt}}\big((\Sigma^{\mathrm{seq}})^{*}\big)\, \rho. \tag{1}$$

Now consider the following event: $\mathtt{B}$ always yields $\mathtt{true}$ (i.e., the loop runs an infinite number of iterations), and in the first n iterations $\mathtt{P}$ has output in the set $A_1 \times \cdots \times A_n$. When ρ' is the state after the first iterations (conditioned on the outputs being in $A_1 \times \cdots \times A_n$, and $\mathtt{B}$ yielding $\mathtt{true}$ in the first n iterations), then the conditional probability that the loop will run an infinite number of iterations (with arbitrary further output) is just $\operatorname{tr} \mathtt{R}^{\mathrm{nt}}\big((\Sigma^{\mathrm{seq}})^{\infty}\big)\rho'$. Writing this as a formula we get the last of our axioms for $\mathtt{R}$:

$$\operatorname{tr} \mathtt{R}^{\mathrm{nt}}\big(A_1 \times \cdots \times A_n \times (\Sigma^{\mathrm{seq}})^{\infty}\big)\, \rho \\ = \operatorname{tr} \mathtt{R}^{\mathrm{nt}}\big((\Sigma^{\mathrm{seq}})^{\infty}\big) \prod_{i=1}^{n} \big(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\big)\,(\rho),$$

which by (1) defines the left hand side.

Note that using these axioms define $\mathtt{R}^{\mathrm{fin}}$ and $\mathtt{R}^{\mathrm{nt}}$ on $A_1 \times \cdots \times A_n$ and $A_1 \times \cdots \times A_n \times (\Sigma^{\mathrm{seq}})^{\infty}$ (for all $n \geq 1$, A_i measurable), i.e., on a set of generators the sigma-algebra of $(\Sigma^{\mathrm{seq}})^{\mathrm{seq}}$. Therefore we can hope that these axioms will define a unique and existent $\mathtt{R}$ (this is positively answered by Lemma 7.2 below). Then we can finally define the while-program by concatenating $\mathtt{R}$'s outputs, i.e., `while (B) P` $:= \operatorname{flatten}(\mathtt{R})$.

Collecting the axioms stated in the above text, we get the following definition:

Definition 7.1 (Loops) Let $\mathtt{P}$ be a program and $\mathtt{B}$ a probability preserving PMVM with outcomes in $\{\mathtt{true}, \mathtt{false}\}$.

Then let R be the probability preserving mixed measurement with outcomes in $(\Sigma^{\mathrm{seq}})^{\mathrm{seq}}$ satisfying

$$\mathtt{R}^{\mathrm{fin}}(A_1 \times \cdots \times A_n)\,\rho = \mathtt{B}(\mathtt{false}) \circ \prod_{i=1}^{n}\bigl(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\bigr)\,(\rho).$$

$$\operatorname{tr}\,\mathtt{R}^{\mathrm{nt}}(A_1 \times \cdots \times A_n)\,\rho = \begin{cases} \operatorname{tr}\,\mathtt{P}^{\mathrm{nt}}(A_n)\ \mathtt{B}(\mathtt{true}) \circ \prod_{i=1}^{n-1}\bigl(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\bigr)\,(\rho), & n \geq 1 \\ 0, & n = 0. \end{cases}$$

$$\operatorname{tr}\mathtt{R}^{\mathrm{nt}}\bigl(A_1 \times \cdots \times A_n \times (\Sigma^{\mathrm{seq}})^{\infty}\bigr)\,\rho = \operatorname{tr}\mathtt{R}^{\mathrm{nt}}\bigl((\Sigma^{\mathrm{seq}})^{\infty}\bigr)\,\prod_{i=1}^{n}\bigl(\mathtt{P}^{\mathrm{fin}}(A_i) \circ \mathtt{B}(\mathtt{true})\bigr)\,(\rho).$$

We then define `while (B) P` $:= \mathrm{flatten}(\mathtt{R})$.

The next lemma tells us that `while (B) P` is in fact a definition.

Lemma 7.2 *Let the situation be as in Definition 7.1. Then* `R` *and* `while (B) P` *always exist, are uniquely defined and* `while (B) P` *is a program.*

The following fact constitutes some evidence that the previous definition in fact complies with the intuitive meaning of a while-program:

Lemma 7.3 (Unrolling `while`-loops) *Let* `P` *and* `B` *be as in Definition 7.1. Then*

$$\mathtt{while\ (B)\ P} \quad = \quad \mathtt{if\ (B)}\ \{\ \mathtt{P;\ while\ (B)\ P}\ \}$$

Of course, `P` is not uniquely described by the equation in this lemma.

We give a simple example of a `while`-loop:

```
var o :  bit;
o := 1;
while (o=1) {
    H₂o; print 1 }
```

This program has a one-bit-variable `o` which is initially initialised to $|1\rangle$. Then its is repeatedly measured in the computational basis, until the outcome does not equal 1. Each time 1 is measured, a H_2-transformation is applied to o and the symbol 1 is output.

Acknowledgements

We thank the anonymous referee and Peter Selinger for helpful comments. This work was founded by the European Project ProSecCo IST-2001-39227.

References

[1] Davies, E. B., "Quantum Theory of Open Systems," Academic Press, London, 1976.

[2] Gay, S. J. and R. Nagarajan, *Communicating quantum processes*, in: P. Selinger, editor, *2nd International Workshop on Quantum Programming Languages*, 2004, pp. 91–107.

[3] Lalire, M. and P. Jorrand, *A process algebraic approach to concurrent and distributed quantum computation: operational semantics*, in: P. Selinger, editor, *2nd International Workshop on Quantum Programming Languages*, 2004, pp. 109–126.

[4] Nielsen, M. and I. Chuang, "Quantum Computation and Quantum Information," Cambridge University Press, Cambridge, 2000.

[5] Preskill, J., *Lecture notes for physics 229: Quantum information and computation* (1998), online available at http://www.theory.caltech.edu/people/preskill/ph229/.

[6] Selinger, P., *Towards a quantum programming language*, Mathematical Structures in Computer Science **14** (2004), pp. 527–586.

[7] Selinger, P. and B. Valiron, *A lambda calculus for quantum computation with classical control*, in: *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications*. Springer LNCS 3461, 2005, pp. 354–368.

[8] Valiron, B., *Quantum typing*, in: P. Selinger, editor, *2nd International Workshop on Quantum Programming Languages*, 2004, pp. 163–178.