

# Definition of the QMC Specification Language

Nikolaos Papanikolaou

October 5, 2008

## 1 Language Description

Various programming and specification formalisms have been proposed in order to address the shortcomings of the quantum circuit model when describing quantum protocols. These include quantum programming languages and quantum process algebras (see [2] for a survey). We have built an imperative-style concurrent specification language for the needs of the quantum model-checking tool QMC. We will proceed to discuss the nature of this specification language, stating its formal syntax and semantics.

The language is designed to allow for the description of systems with classical data and communication as well as manipulation and transmission of a finite number of qubits. A QMC model consists of: (1) a set of process declarations, and (2) a set of global (channel) variable declarations.

Each process defines the behaviour of a system component and has its own local variables (classical variables of integer, real or boolean type, and qubit variables) as well as access to all global variables (which represent channels of communication between processes).

The minimal QMC program consists of a single process with no actions:

```
program Minimal;
2
process SingleProcess;
begin
4
end;
6
endprogram.
8
```

Classical variables are declared and used just as in conventional computer languages such as Pascal [4]; the language includes assignment statements and simple expressions (e.g.  $a := b + 2$ ; is a valid statement assuming  $a$ ,  $b$  are suitably declared). Qubit variables store references to qubits in the global quantum state and are used in one of four ways (assuming  $a$ ,  $q$  have been declared as qubit variables):

- the statement  $q := \text{newqubit}$ ; allocates a new qubit in the global quantum state and sets the value of  $q$  to represent the index of that qubit. Before such a statement is issued, although a qubit variable  $q$  may have been declared it does not stand for an actual qubit in the quantum state.
- the statement  $a := \text{savequbit } q$ ; stores the global quantum state at that point in execution. The ability to create *history variables* such as  $a$  is a special feature of the language which is useful for property verification.

- in statements which apply quantum operators ( $H$ ,  $CNot$ ,  $S$ ,  $X$ ,  $Y$ ,  $Z$ ) to particular qubits, the corresponding qubit variables are used, e.g.:  $\text{had } q$ ;  $\text{X } q$ ;  $\text{cnot } a \ q$ ;
- Measurements of qubits in the standard basis are performed using expressions of the form  $\text{meas } q$ . Normally these expressions are assigned to an integer value, so that the measurement outcome (0 or 1) is recorded.

Apart from assignments and the statements of the forms discussed above, processes are able to send and receive the values of variables on channels. The sample program below demonstrates this feature of the language. Note that the syntax for sending and receiving is reminiscent of the formalism CSP [3].

```
program SendReceive;
var ch: channel of integer;
2

process Sender;
4
var a: integer;
begin
6
    a := 2;
    ch!a;
8
end;
10

process Receiver;
var b: integer;
12
begin
    ch?b;
14
end;
16
endprogram.
```

This example also demonstrates two other characteristics of the language: *parallelism* and *executability*. The `SendReceive` system model declares one channel variable and two parallel processes `Sender` and `Receiver`. All processes in QMC are supposed to be executed in parallel, leading to several possible interleavings. However, the action `ch?b` cannot be executed before the channel `ch` has received a value, so there is only possible execution of this particular program. There is no explicit parallelism operator for processes (which is common in other specification languages), as it is implicit.

Simple statements can be combined together to form *blocks*, which are executed in a single step. A block is simply a sequence of statements enclosed in braces (`{` and `}`). The semantics of the language is such that one step in the execution of a process consists of either a single statement by itself or a block.

QMC models can also include non-deterministic choices and loops. In particular, a process can perform a choice between

several sequences of statements, and such a choice can be performed repeatedly. Here is an example of a process involving a loop (this loop is repeated up to a maximum number of times, which is implementation-dependent):

```

process Looping;
var a: integer; q: qubit;           2
begin
  q := newqubit;                     4
  do
    □ a:=0; had q;                     6
    □ a:=1; X q;
  od                                   8
end;

```

The Looping process repeatedly chooses, either to set a to 0 and then apply the  $H$  gate to q, or to set a to 1 and then apply the  $X$  gate to q. The first statement in an option (an option is a sequence of statements preceded by the symbol ‘□’ – in ASCII form this is represented by ‘:’) can be an expression stating a condition; this condition is used to determine whether the option is executable or not. These conventions are habitual in guarded-command languages and are inspired by a proposal of Dijkstra [1].

Next we define the formal syntax of QMC’s specification language and the abstract machine which defines its operational semantics.

## 2 Syntax

The BNF grammar in Figure 1.2 defines in turn: expressions ( $e$ ), statements ( $S$ ), guarded commands ( $G$ ), variable declarations ( $V$ ), process declarations ( $P$ ) and model definitions or programs ( $M$ ).

**Figure 1** Concrete Syntax for QMC’s specification language.

```

t ::= integer | bool | real | qubit
   | channel of t
e ::= n | r | x | e1 + e2 | e1 - e2 | e1 * e2 | e1/e2
   | true | false | not e | e1 and e2 | e1 or e2
   | e1 = e2 | e1 < e2 | e1 > e2 | meas x̃ | newqubit
S ::= e | x := e | x1!x2 | x1?x2 | cnot x1x2 | had x
   | ph x | X x | Y x | Z x | S1;S2
H ::= □SH | □S
G ::= if H fi | do H od
C ::= S; | G | C1 C2 | ε
V ::= var D | ε
D ::= x : t; D | ε
P ::= process p V begin C end P | ε
M ::= program p V begin P end

```

The semantics of the language will be defined over the *abstract syntax* of the language, given in Figure 2.

**Figure 2** Abstract Syntax for QMC’s specification language.

```

t ::= integer | bool | real | qubit
   | channel of t
e ::= n | r | x | e1 + e2 | e1 - e2 | e1 * e2 | e1/e2
   | true | false | ¬ e | e1 ∧ e2 | e1 ∨ e2
   | e1 = e2 | e1 < e2 | e1 > e2 | meas x̃ | newqubit
S ::= e | x := e | x1!x2 | x1?x2 | cnot x1x2 | had x
   | ph x | X x | Y x | Z x | S1;S2
H ::= □SH | □S
G ::= if H | do H
C ::= S; | G | C1 C2 | ε
D ::= x : t; D | ε
P ::= D C P | ε
M ::= D P

```

## 3 Configurations and Notational Conventions

A configuration of the abstract machine used for the definition of QMC semantics consists of a *global component*, a pair  $(\psi, \sigma, D)$  and a set of *local components* of the form  $(\gamma, P)$ . The global component comprises a quantum state  $\psi$  and a classical store  $\sigma$ , which assigns types and values to globally declared variables (channels). Each local component corresponds to the state of an individual process declared in a particular model.

So the configuration of the abstract machine in its most general form is given by a tuple

$$((\psi), \sigma, D; (\gamma_1, P_1) \parallel \cdots \parallel (\gamma_n, P_n))$$

The effect of a statement in QMC may have an effect on the global state, an effect on the local state of a process, or both. When defining the semantics of statements that affect only the state of a process, we use *partial configurations* of the form:

$$((\psi), \sigma; \gamma, C)$$

Partial configurations of the form  $((\psi), \sigma; \gamma)$ , in which there is no syntactic term remaining, are *terminal*.

## 4 Transition Relations

We define one transition relation for each category of syntax, in particular, a relation  $\rightarrow_N$  for each nonterminal  $N$  in the grammar of the language. Thus we have:

- for expressions, a relation  $\rightarrow_e$
- for statements, a relation  $\rightarrow_S$
- for options (non-deterministic choices), a relation  $\rightarrow_H$
- for guarded commands, a relation  $\rightarrow_G$

- for general commands, a relation  $\rightarrow_C$
- for variable declarations, a relation  $\rightarrow_D$
- for processes, a relation  $\rightarrow_P$
- for programs, a relation  $\rightarrow_M$

## 5 Operational Semantics

Assigning an expression  $e$  to a variable  $x$  simply updates the local variable store  $\gamma$  with the value of  $e$ .

$$\frac{(|\psi\rangle, \sigma; \gamma, e) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, x := e; C) \rightarrow_S (|\psi\rangle, \sigma; \gamma', C)} \quad (\text{R-ASSN})$$

where  $\gamma' = \gamma[x \mapsto \text{val}]$

Sending the value of a variable  $x_2$  on a channel  $x_1$  has the effect of updating the global store  $\sigma$  (since channels are always considered global variables):

$$\frac{}{(|\psi\rangle, \sigma; \gamma, x_1!x_2; C) \rightarrow_S (|\psi\rangle, \sigma'; \gamma', C)} \quad (\text{R-SEND})$$

if  $\sigma_v(x_1) = \text{null}$  where  $\sigma' = \sigma[x_1 \mapsto \gamma_v(x_2)]$   
and  $\gamma' = \begin{cases} \gamma[x_2 \mapsto \text{null}] & \text{if } \gamma_t(x_2) = \text{qubit} \\ \gamma & \text{otherwise} \end{cases}$

The rule for receiving a value from a channel is in the same style:

$$\frac{}{(|\psi\rangle, \sigma; \gamma, x_1?x_2; C) \rightarrow_S (|\psi\rangle, \sigma'; \gamma', C)} \quad (\text{R-RECEIVE})$$

if  $\sigma_v(x_1) \neq \text{null}$  where  $\sigma' = \sigma[\gamma(x_1) \mapsto \text{null}]$   
and  $\gamma' = \gamma[x_2 \mapsto \sigma_v(x_1)]$

Next are the rules stating the effect of statements **had**  $x$ , **cnot**  $x_1 x_2$ , **ph**  $x$ , **X**  $x$ , **Y**  $x$ , **Z**  $x$ , all of which are in the same vein. The notation  $\text{val.index}$  is used to represent the index of the qubit (with reference to the global quantum state  $|\psi\rangle$ ) represented by  $\text{val}$ .

$$\frac{(|\psi\rangle, \sigma; \gamma, x) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, \text{had } x; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-HAD})$$

where  $|\psi'\rangle = H_{\text{val.index}}|\psi\rangle$

$$\frac{(|\psi\rangle, \sigma; \gamma, x_1) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val}_1), \quad (|\psi\rangle, \sigma; \gamma, x_2) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val}_2)}{(|\psi\rangle, \sigma; \gamma, \text{cnot } x_1 x_2; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-CNOT})$$

where  $|\psi'\rangle = C_{\text{val}_1.\text{index}, \text{val}_2.\text{index}}|\psi\rangle$

$$\frac{(|\psi\rangle, \sigma; \gamma, x) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, \text{ph } x; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-APPLY S})$$

where  $|\psi'\rangle = S_{\text{val.index}}|\psi\rangle$

$$\frac{(|\psi\rangle, \sigma; \gamma, x) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, \text{X } x; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-APPLY X})$$

where  $|\psi'\rangle = X_{\text{val.index}}|\psi\rangle$

$$\frac{(|\psi\rangle, \sigma; \gamma, x) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, \text{Y } x; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-APPLY Y})$$

where  $|\psi'\rangle = Y_{\text{val.index}}|\psi\rangle$

$$\frac{(|\psi\rangle, \sigma; \gamma, x) \rightarrow_e (|\psi\rangle, \sigma; \gamma, \text{val})}{(|\psi\rangle, \sigma; \gamma, \text{Z } x; C) \rightarrow_S (|\psi'\rangle, \sigma; \gamma, C)} \quad (\text{R-APPLY Z})$$

where  $|\psi'\rangle = Z_{\text{val.index}}|\psi\rangle$

The final rule for the transition relation  $\rightarrow_S$  simply defines the effect of sequencing.

$$\frac{(|\psi\rangle, \sigma; \gamma, S_1) \rightarrow_S (|\psi'\rangle, \sigma'; \gamma')}{(|\psi\rangle, \sigma; \gamma, S_1; S_2) \rightarrow_S (|\psi'\rangle, \sigma'; \gamma', S_2)} \quad (\text{R-SSEQ})$$

Non-deterministic choices consists of several options (sequences of statements) each of which may or may not be executable. Assuming only executable statements we have the following rules for transition relation  $\rightarrow_H$ .

$$\frac{(|\psi\rangle, \sigma; \gamma, S) \rightarrow_S (|\psi'\rangle, \sigma'; \gamma', S')}{(|\psi\rangle, \sigma; \gamma, \square S H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', S')} \quad (\text{R-OP}_1)$$

$$\frac{(|\psi\rangle, \sigma; \gamma, S) \rightarrow_S (|\psi'\rangle, \sigma'; \gamma', S')}{(|\psi\rangle, \sigma; \gamma, \square S) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', S')} \quad (\text{R-OP}_2)$$

$$\frac{}{(|\psi\rangle, \sigma; \gamma, \square S H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', H)} \quad (\text{R-OP}_3)$$

$$\frac{(|\psi\rangle, \sigma; \gamma, H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', H'), \quad (|\psi'\rangle, \sigma'; \gamma', H') \rightarrow_H (|\psi''\rangle, \sigma''; \gamma'', H'')}{(|\psi\rangle, \sigma; \gamma, H) \rightarrow_H (|\psi''\rangle, \sigma''; \gamma'', H'')} \quad (\text{R-OP}_4)$$

Non-deterministic choices are always enclosed in a *guarded command* of the form **if** ... **fi** or of the form **do** ... **od**. The first of these forms reduces to a single  $\rightarrow_H$  transition, while the second form is a little more involved, as it involves repetition.

$$\frac{(|\psi\rangle, \sigma; \gamma, H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', S)}{(|\psi\rangle, \sigma; \gamma, \text{if } H) \rightarrow_G (|\psi'\rangle, \sigma'; \gamma', S)} \quad (\text{R-GC}_1)$$

We define  $\gamma = (\kappa, \tau, i)$  where  $i$  is the *iteration counter* for a process; the value of  $i$  is always in the range  $\{1, IMAX\}$  where  $IMAX$  is an implementation-dependent constant representing the maximum number of times that a loop is repeated. In previous rules we have used the notations  $\gamma_v \equiv \kappa$ ,  $\gamma_t \equiv \tau$ , for extracting the value and type of a given variable from the store  $\gamma$  (analogously  $\sigma_v$  and  $\sigma_t$  have been used to extract value and type information from the global store  $\sigma$ ).

As long as the iteration counter for a particular process is less than  $IMAX$  while executing a loop, the content of the loop is repeated:

$$\frac{(|\psi\rangle, \sigma; \gamma, H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', S), \quad i < IMAX}{(|\psi\rangle, \sigma; \gamma, \text{do } H) \rightarrow_G (|\psi'\rangle, \sigma'; \gamma', S \text{ do } H)} \quad (\text{R-GC}_2)$$

where  $\gamma' = (\kappa, \tau, i) \Rightarrow \gamma'' = (\kappa, \tau, i + 1)$

Once the maximum number of iterations is reached, the loop ends:

$$\frac{(|\psi\rangle, \sigma; \gamma, H) \rightarrow_H (|\psi'\rangle, \sigma'; \gamma', S), \quad i = \text{IMAX}}{(|\psi\rangle, \sigma; \gamma, \mathbf{do} H) \rightarrow_G (|\psi'\rangle, \sigma'; \gamma''', S)} \quad (\text{R-GC}_3)$$

where  $\gamma' = (\kappa, \tau, i) \implies \gamma''' = (\kappa, \tau, 0)$

The following rules define the general transition relation  $\rightarrow_C$ :

$$\frac{(|\psi\rangle, \sigma; \gamma, S) \rightarrow_S (|\psi'\rangle, \sigma'; \gamma', S')}{(|\psi\rangle, \sigma; \gamma, S) \rightarrow_C (|\psi'\rangle, \sigma'; \gamma', S')} \quad (\text{R-COM}_1)$$

$$\frac{(|\psi\rangle, \sigma; \gamma, GC) \rightarrow_G (|\psi'\rangle, \sigma'; \gamma', S')}{(|\psi\rangle, \sigma; \gamma, GC) \rightarrow_C (|\psi'\rangle, \sigma'; \gamma', S')} \quad (\text{R-COM}_2)$$

$$\frac{(|\psi\rangle, \sigma; \gamma, C_1) \rightarrow_C (|\psi'\rangle, \sigma'; \gamma', C'_1)}{(|\psi\rangle, \sigma; \gamma, C_1 C_2) \rightarrow_C (|\psi'\rangle, \sigma'; \gamma', C'_1 C_2)} \quad (\text{R-COM}_3)$$

The following rules define the semantics of variable declarations, which are either local or global (and hence affect either  $\gamma$  or  $\sigma$  accordingly).

$$\frac{(|\psi\rangle, \sigma; \gamma, x : t; D) \rightarrow_D (|\psi\rangle, \sigma'; \gamma, D)}{\text{where } \sigma' = \sigma \cup \{(x \mapsto \text{null}, t)\}} \quad (\text{R-VDEC}_1)$$

if  $t$  = channel type and  $x$  is undefined in  $\sigma$

$$\frac{(|\psi\rangle, \sigma; \gamma, x : t; D C) \rightarrow_D (|\psi\rangle, \sigma; \gamma', D C)}{\text{where } \gamma' = \begin{cases} \gamma \cup \{(x \mapsto 0, \text{int})\} & \text{if } t = \mathbf{integer} \\ \gamma \cup \{(x \mapsto 0.0, \text{real})\} & \text{if } t = \mathbf{real} \\ \gamma \cup \{(x \mapsto \text{false}, \text{bool})\} & \text{if } t = \mathbf{bool} \\ \gamma \cup \{(x \mapsto \text{null}, \text{qubit})\} & \text{if } t = \mathbf{qubit} \end{cases}} \quad (\text{R-VDEC}_2)$$

if  $t \neq$  channel and  $x$  undefined in  $\gamma$

$$\frac{(|\psi\rangle, \sigma, D; P) \rightarrow_D (|\psi\rangle, \sigma', D'; P)}{(|\psi\rangle, \sigma', D; P) \rightarrow_D (|\psi\rangle, \sigma'', P)} \quad (\text{R-VDEC}_3)$$

$$\frac{(|\psi\rangle, \sigma; \gamma, D C) \rightarrow_D (|\psi\rangle, \sigma'; \gamma', D' C)}{(|\psi\rangle, \sigma; \gamma', D' C) \rightarrow_D (|\psi\rangle, \sigma; \gamma'', C)} \quad (\text{R-VDEC}_4)$$

### Process Transitions

$$\frac{(|\psi\rangle, \sigma; \gamma_i, D_i C_i) \rightarrow_D (|\psi\rangle, \sigma; \gamma'_i, C_i) \text{ for all } i}{(|\psi\rangle, \sigma; (\gamma_1, D_1 C_1) \parallel \dots \parallel (\gamma_n, D_n C_n)) \rightarrow_P (|\psi\rangle, \sigma; (\gamma'_1, C_1) \parallel \dots \parallel (\gamma'_n, C_n))} \quad (\text{R-PROC}_1)$$

$$\frac{(|\psi\rangle, \sigma; \gamma_i, C_i) \rightarrow_C (|\psi'\rangle, \sigma'; \gamma'_i, C'_i)}{(|\psi\rangle, \sigma; (\gamma_1, C_1) \parallel \dots \parallel (\gamma_i, C_i) \parallel \dots \parallel (\gamma_n, C_n)) \rightarrow_P (|\psi'\rangle, \sigma'; (\gamma_1, C_1) \parallel \dots \parallel (\gamma'_i, C'_i) \parallel \dots \parallel (\gamma_n, C_n))} \quad (\text{R-PROC}_2)$$

### Program Transitions

$$\frac{(|\psi\rangle, \sigma, D; P) \rightarrow_D (|\psi\rangle, \sigma'; P)}{(|\psi\rangle, \sigma, D; P) \rightarrow_M (|\psi\rangle, \sigma'; P)} \quad (\text{R-PROG}_1)$$

$$\frac{(|\psi\rangle, \sigma, ; P) \rightarrow_P (|\psi'\rangle, \sigma'; P')}{(|\psi\rangle, \sigma, ; P) \rightarrow_M (|\psi'\rangle, \sigma'; P')} \quad (\text{R-PROG}_2)$$

## References

- [1] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [2] Simon J. Gay. Quantum programming languages: Survey and bibliography. *Mathematical Structures in Computer Science*, 16(4), 2006.
- [3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [4] Niklaus Wirth. The programming language Pascal. *Acta Informatica*, 1:35–63, 1971.