

Quantum Informatics: **Algorithms and Languages**

Roland Rüdiger

Festkolloquium in honor of Rüdiger Göbel
University of Duisburg-Essen, April 24-25, 2009

Abstract

Quantum Informatics is a new research area which deals with traditional notions of classical informatics in the situation that the carrier of information is a physical system with genuine quantum properties. Shor's algorithm for efficiently factorizing large integers is still the most prominent among many other algorithms, which have been found so far. After a brief discussion of some physical preliminaries, this algorithm will be presented in some detail. Formulating algorithms requires some sort of notation. Pseudo-code and conventional mathematical notations are suitable for persons but, in general, not sufficiently concise for the use by automata (compiler). Therefore, quantum programming languages (QPLs) have been designed, which support the development of quantum algorithms and quantum programs. In combination with a classical simulator they can be used to run quantum programs on a conventional PC, although usually not efficiently. The talk concludes with a brief survey on QPLs in general and a detailed discussion of a specific example.

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Qu-Informatics vs Qu-Information Theory

1/19

- Common basis of both fields: the physical carrier of information is a quantum system
- *Quantum Information Theory*: deals with the notion of “information” as conceived by Shannon
- *Quantum Informatics*: deals with traditional concepts of informatics such as algorithms, automata, programming languages, . . .

What's new about Quantum Informatics?

2/19

- Doubts have been cast on a fundamental thesis of theoretical informatics:
- The **Strong Church-Turing Thesis**
(as formulated by Nielsen / Chuang [NC00, p. 140]):
“Any model of computation can be simulated on a probabilistic Turing machine with at most a polynomial increase in the number of elementary operations required”.
- The claimed model independence of computational complexity appears not to be true.
- The expected speed-up in executing algorithms on quantum systems seems to falsify this thesis.

Outline

- 1 Introduction
 - Quantum Informatics
 - **Some Terminology in a Nutshell**
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

A Quick Tour of Qu-Mechanics (pt 1)

3/19

- The arena (in theory) of quantum processes is a complex vector space with an inner product (**Hilbert-space**) \mathcal{H} .
- **State of a quantum system**:
a positive operator $\rho : \mathcal{H} \rightarrow \mathcal{H}$ with $\text{tr } \rho = 1$.
- **Pure state**: a state with “as little randomness as possible”, which is extremal. This can be written as $\rho = |\psi\rangle\langle\psi|$ with the **state-vector** $|\psi\rangle \in \mathcal{H}$ and its dual $\langle\psi| \in \mathcal{H}^*$.
- **Observable**: traditionally, a self-adjoint operator A , $A = A^*$ or, more generally, a **Positive Operator Valued Measure (POVM)**: a set of positive operators $\{M_x\}$ with $\sum_{x \in X} M_x = \mathbb{1}$,
 $X =$ set of possible measuring outcomes.
- **Probability** of getting x : $p_x = \text{tr } \rho M_x$.

A Quick Tour of Qu-Mechanics (pt 2)

4/19

- Allowed operations T on states (“**quantum channels**”) are those which preserve the defining properties of states: $\rho \geq 0$ and $\text{tr } \rho = 1$.
Mathematically, these are **completely positive (cp) maps**: for any positive T , $T \otimes \mathbb{I}_n$ is positive for all $n \in \mathbb{N}$.
- Kraus representation**: the standard form for channels:
 $\rho' = \sum_k K_k \rho K_k^*$ with $\sum_k K_k^* K_k = \mathbb{1}$.
- ex.: $\rho' = \sum_k p_k U_k \rho U_k^*$ with unitaries U_k , $\sum_k p_k = 1$, $p_k \geq 0$
- Preparation / measurement**: the action of encoding / extracting classical information into / from a quantum system.
- In a **closed system**, the **time evolution** is **reversible**:
 $\rho' = U \rho U^*$, for pure states: $|\psi'\rangle = U|\psi\rangle$, with a unitary U .

Quantum Mechanics—a Weird Theory?

5/19

- Physical phenomena can be extremely strange in spite of the fact that the mathematical description is seemingly simple.
- Example: Superposition of probability amplitudes (addition of state vectors)
- One of Einstein's arguments that Quantum Theory, although successful, is a preliminary theory: it does not describe physical phenomena as processes evolving in space and time.

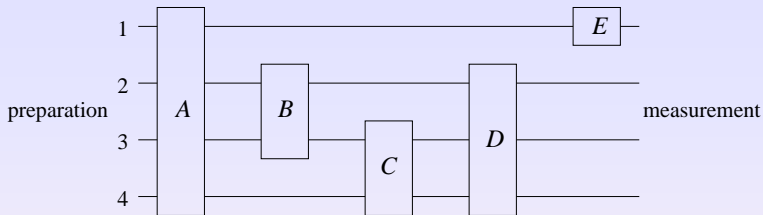
Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 **The Gate Model**
 - **Boolean Networks Quantized**
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Quantum Circuit

6/19

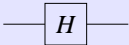
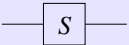
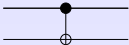
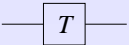
- An example of a quantum circuit, consisting of several unitaries



- $U = (E_1 \otimes 1 \otimes 1 \otimes 1)(1 \otimes D_{234})(1 \otimes 1 \otimes C_{34})(1 \otimes B_{23} \otimes 1)(A_{1234})$
- There are finite sets of universal elementary gates which can be used as building blocks for more complex operations.
- Universal elementary sets *approximate* unitary operations.

Standard Set of Universal Gates

7/19

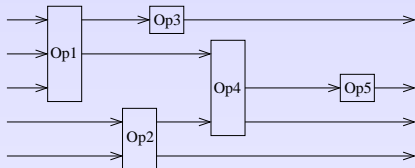
| Hadamard = Mix | Phase | CNOT = XOR | T-gate |
|---|---|---|---|
|  |  |  |  |
| $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ | $\begin{pmatrix} 1 & \\ & i \end{pmatrix}$ | $\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & & & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & \\ & e^{i\pi/4} \end{pmatrix}$ |
| $ x\rangle \rightarrow 2^{-1/2} \left((-1)^x x\rangle + 1-x\rangle \right)$ | $ x\rangle \rightarrow e^{i\frac{\pi}{2}x} x\rangle$ | $ x\rangle \otimes y\rangle \rightarrow x\rangle \otimes x \oplus y\rangle$ | $ x\rangle \rightarrow e^{i\frac{\pi}{4}x} x\rangle$ |

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - **The General Model**
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Quantum Circuits as Directed Acyclic Graphs

8/19



- Some situations cannot adequately be described by the standard circuit model, e.g.
 - time evolution, prepar., and measurements interleaved
 - computation of probabilistic functions (“**subroutine problem**”)
- A general model, as proposed by Aharonov, Kitaev, and Nisan: **A quantum circuit is a Directed Acyclic Graph.**
- **Nodes** represent general quantum operations (channels), **edges** (“quantum wires”) represent the (consistent) sequential composition of these mappings.

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 **Quantum Algorithms**
 - **Speed-up by “quantum hardware”**
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Quantum Parallelism, Classical Simulatability

9/19

- A “Folk Theorem”: The source of speed-up by quantum systems is “quantum parallelism”, i.e. . . .
- . . . applying a quantum operation to the state $|\psi\rangle = \sum_{k=0}^{d-1} a_k |k\rangle$ of a n -qubit system, where $d = 2^n$, means to act simultaneously on 2^n integers in one step.
- The question in a more concise form: Which quantum systems can / cannot be simulated efficiently by classical systems?
- Presently, only partial answers are known, see Jozsa and Miyake [Joz08], [JM08] for details.

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - **Integer Factorization**
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Shor's Algorithm (simplified form)

10/19

- **The Problem:** Given a composite $N \in \mathbb{N}$, determine a prime factor of N
- **The Algorithm:**
 - Choose a random $a \in \mathbb{Z}_N$, $a \geq 2$.
 - Determine $r = \text{ord}_N a$, the order of a modulo N (least r with $a^r \equiv 1 \pmod{N}$), the period of $f_{a,N}(x) = a^x \pmod{N}$:
 - If r is even, factorize $a^r - 1$ into:
$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \pmod{N}$$
 - Therefore: The factors of N must also be factors of $N_- := a^{r/2} - 1$ and / or $N_+ := a^{r/2} + 1$.
 - Determine (efficiently) $d_- = \text{gcd}(N, N_-)$ and $d_+ = \text{gcd}(N, N_+)$. These are factors of N .
 - A complication: The factors might be trivial:
$$1 \leq d_- < N, 1 < d_+ \leq N$$

Shor's Algorithm (complete form)

11/19

(in "CLRS-style" pseudocode: "CLRS" refers to: *Introduction to Algorithms* by Cormen, Leiserson, Rivest, Stein)FACTORIZE(N)

```

1  if  $N$  is even
2    then return  $(2, N/2)$ 
3  if  $N = q^b$  for prime  $q \geq 3$  and  $b \geq 2$ 
4    then return  $(q, N/q)$ 
5  repeat


---


6    repeat choose  $a \in \mathbb{Z}_N, a \geq 2$ 
7       $d \leftarrow \text{gcd}(a, N)$ 
8      if  $d > 1$ 
9        then return  $(d, N/d)$ 
10      $r \leftarrow \text{FIND-ORDER}_N(a)$ 
11     until no failure indicated and  $r$  is even


---


12      $d_+ \leftarrow \text{gcd}(N, a^{r/2} + 1)$ 
13   until  $d_+ < N$ 
14    $d_- \leftarrow \text{gcd}(N, a^{r/2} - 1)$ 
15   return  $(d_+, d_-)$ 

```

\triangleright we luckily guessed d correctly
 \triangleright the quantum part, $x \mapsto a^x \bmod N$
 \triangleright evaluate 2^{nd} argument modulo N
 \triangleright the algorithm guarantees $1 < d_+, d_- < N$

Quantum Algorithm Order-Finding

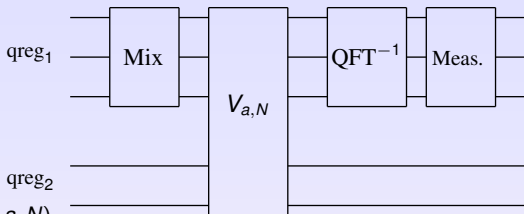
12/19

FIND-ORDER_N(*a*)

```

1  repeat RESET
2    MIX(qreg1)
3     $V_{a,N}(qreg_1 \otimes qreg_2)$ 
4    QFT-1(qreg1)
5     $m \leftarrow$  MEASURE(qreg1)
6  until  $m \neq 0$ 
7   $t \leftarrow$  length[qreg1]
8   $r \leftarrow$  TEST-CONVERGENTS( $2^t, m, a, N$ )
9  return r
10 ▷  $r = -1$  indicates failure
11 ▷ otherwise r is guaranteed to be the order of a modulo N

```



- TEST-CONVERGENTS determines the convergents p_i/q_i of $m/2^t$ and returns the smallest q_i with $a^{q_i} \equiv 1 \pmod{N}$ if such q_i exists; otherwise, it indicates an error.

The Algorithm in Action

13/19

- Example: $N = 53 \cdot 101 \cdot 157 = 840\,421$

- excerpt:

| $a \in \mathbb{Z}_N$ | r | N_+ | N_- | d_+ | d_- | |
|----------------------|------|--------|--------|--------|-------|-----------------------|
| ... | | | | | | |
| 784 | 650 | 507424 | 507422 | 15857 | 53 | |
| 785 | — | — | — | 157 | — | chance hit |
| 786 | 325 | — | — | — | — | r odd |
| 787 | 1300 | 108173 | 108171 | 8321 | 101 | |
| 788 | 1950 | 615597 | 615595 | 157 | 5353 | |
| 789 | 650 | 507424 | 507422 | 15857 | 53 | |
| 790 | 3900 | 840421 | 840419 | 840421 | 1 | trivial factorization |
| 791 | 780 | 615597 | 615595 | 157 | 5353 | |
| 792 | 1300 | 108173 | 108171 | 8321 | 101 | |
| 793 | 1300 | 840421 | 840419 | 840421 | 1 | trivial factorization |
| 794 | 390 | 332999 | 332997 | 53 | 15857 | |
| 795 | — | — | — | 53 | — | chance hit |
| 796 | 3900 | 732250 | 732248 | 101 | 8321 | |
| 797 | 3900 | 224826 | 224824 | 5353 | 157 | |
| 798 | 156 | 224826 | 224824 | 5353 | 157 | |
| 799 | 650 | 332999 | 332997 | 53 | 15857 | |
| 800 | 3900 | 840421 | 840419 | 840421 | 1 | trivial factorization |
| 801 | 1300 | 732250 | 732248 | 101 | 8321 | |
| 802 | 390 | 332999 | 332997 | 53 | 15857 | |
| 803 | 3900 | 108173 | 108171 | 8321 | 101 | |
| 804 | 1950 | 332999 | 332997 | 53 | 15857 | |
| 805 | 3900 | 507424 | 507422 | 15857 | 53 | |
| ... | | | | | | |

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - **Designing Quantum Programming Languages (QPLs)**
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

Are We Putting the Cart Before the Horse?

14/19

The argument: presently there are no quantum computers of a reasonable size. So, what are QPLs good for?

A few possible answers:

- Building a computer (the hardware) is only half of the story: computers must be programmable.
- Formulating algorithms requires some sort of notation.
- Pseudocode, although important, is not sufficient.
- Perform experiments with systems (language / simulator), to get some intuition for innovative hardware and . . .
- . . . experiment with so-called “higher order structures”.
- Also useful when designing protocols in quantum cryptography.

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - **The Zoo of QPLs—Tabular survey**
 - A closer look at one QPL

Some Languages and Formalized Notations

15/19

| Language | Author(s) | imperative language | functional language | operational semantic | denotational semantic | Comments |
|-------------------|-----------------------------------|---------------------|---------------------|----------------------|-----------------------|---------------------------------------|
| <i>QCL</i> | Ömer | × | | | | an elaborate QPL |
| <i>qGCL</i> | Sanders/Zuliani | × | | × | | refinement calculus |
| <i>Q language</i> | Bettelli/Calarco/Serafini | × | | | | C++ and C++-based library |
| <i>LanQ</i> | Mlnařik | × | | × | | interprocess communication |
| <i>NDQJava</i> | Xu/ Song/ Qian/ Dai/ Zhang | × | | | | Java-embedded QPL |
| | Sabry | | | × | | Haskell as a QPL |
| <i>QPL/QFC</i> | Selinger | | | × | × | first order QPL |
| <i>cQPL</i> | Mauerer | | | × | × | extension of Selinger's <i>QPL</i> |
| <i>QML</i> | Altenkirch/Grattage | | | × | × | first order QPL, linear logic |
| λ_q | van Tonder | | | × | × | λ -calculus |
| | Selinger/Valiron | | | × | × | higher order QPL, λ -calculus |
| | Arrighi/Dowek | | | | × | λ -calculus, linearity |
| <i>QPAIg</i> | Lalire/Jorrand | | | | × | process calculus |
| <i>CQP</i> | Gay/Nagarajan | | | | × | process calculus |

QCL to be explained in more detail.

Some Languages and Formalized Notations

15/19

| Language | Author(s) | imperative language | functional language | operational semantic | denotational semantic | First published |
|-------------------|-----------------------------------|---------------------|---------------------|----------------------|-----------------------|-----------------|
| <i>QCL</i> | Ömer | × | | | | 1998 |
| <i>qGCL</i> | Sanders/Zuliani | × | | × | | 2000 |
| <i>Q language</i> | Bettelli/Calarco/Serafini | × | | | | 2003 |
| <i>LanQ</i> | Mlnařík | × | | × | | 2006 |
| <i>NDQJava</i> | Xu/ Song/ Qian/ Dai/ Zhang | × | | | | 2008 |
| | Sabry | | × | | | 2003 |
| <i>QPL/QFC</i> | Selinger | | × | | × | 2004 |
| <i>cQPL</i> | Mauerer | | × | | × | 2005 |
| <i>QML</i> | Altenkirch/Grattage | | × | | × | 2005 |
| λ_q | van Tonder | | × | | × | 2003 |
| | Selinger/Valiron | | × | × | | 2005 |
| | Arrighi/Dowek | | | × | | 2005 |
| <i>QPAIg</i> | Lalire/Jorrand | | | × | | 2004 |
| <i>CQP</i> | Gay/Nagarajan | | | × | | 2005 |

QCL to be explained in more detail.

Outline

- 1 Introduction
 - Quantum Informatics
 - Some Terminology in a Nutshell
- 2 The Gate Model
 - Boolean Networks Quantized
 - The General Model
- 3 Quantum Algorithms
 - Speed-up by “quantum hardware”
 - Integer Factorization
- 4 Quantum Programming Languages
 - Designing Quantum Programming Languages (QPLs)
 - The Zoo of QPLs—Tabular survey
 - A closer look at one QPL

QCL (Ömer) (pt 1)

16/19

- An imperative language:
“imperative” means: A computation is considered as a sequence of transitions of the program state.
- Features of *QCL*:
 - A procedural language in the tradition of Pascal and C.
 - An excellent choice for teaching QPLs.
 - Embodies a classical sublanguage, which is sufficiently rich for simple programming tasks.
 - Operators are static language constructs, syntactically on the level of procedures.
 - 2 quantum operator types: general unitarian (`operator`) and unitaries related to classical Boolean functions (`qufunct`).

QCL (Ömer) (pt 2)

17/19

- Features of *QCL* (cont'd):
 - Call hierarchy **procedure** → **operator** → **qfunct** → **function**
 - Language construct for inverting operators.
 - Various quantum data types (qubit registers)
 - Functions to manipulate quantum registers.
 - Quantum memory management allowing for local quantum variables.
 - Bennett-style “uncomputation” of auxiliary registers (“ancillas”)

QCL Example (by Ömer), modified

18/19

Shor's algorithm implemented (excerpt)

```

procedure shor(int N) {
  int L = ceil(log(N,2));
  int t = ceil(2*log(N,2));
  qureg reg1[t];
  qureg reg2[L];

  ... some declarations of int variables
  and tests:
  e.g. do we have a trivial N?
  ...

  {
    a = floor(random()*(N-3)) + 2;
    d = gcd(a, N);
    if d > 1 {
      print "Factor =", d;
      print "Nothing to do for the QC.";
      exit;
    }
  }

```

```

/*-----*/
{ reset;
  H(reg1);
  expn(a, N, reg1, reg2);
  measure reg2;
  !dft(reg1);
  measure reg1, m;
} until m != 0;
r = testConvergents(2^t, m, a, N);
/*-----*/

} until (r != -1) and (r mod 2 == 0);
Nplus = powmod(a, r/2, N);
dplus = gcd(N, Nplus + 1);
} until dplus < N;
print "dplus", dplus;
}

```

- Quantum Physics / Quantum Information:

- | | |
|--|--|
| <ul style="list-style-type: none">● Bruß / Leuchs [BL07]● Hirvensalo [Hir01]● Holevo / Werner [HW] | <ul style="list-style-type: none">● Mermin [Mer07]● Nielsen / Chuang [NC00]● Stolze / Suter [SS04] |
|--|--|

- Quantum Algorithms:

- Childs / van Dam [CvD08]
- Mosca [Mos08]

- Quantum Programming Languages:

- Gay [Gay05, Gay06]
- Rüdiger [Rüd03, Rüd07, Rüd09]
- Selinger [Sel04a, Sel04b, Sel05, Sel06]
- Sofge [Sof08]
- Unruh [Unr06]

References

References

- [BL07] Dagmar Bruß and Gerd Leuchs, editors. *Lectures on Quantum Information*. Wiley VCH, Weinheim, 2007.
- [CvD08] Andrew M. Childs and Wim van Dam. Quantum algorithms for algebraic problems. Available at <http://arXiv:0812.0380v1>, December 2008.
- [Gay05] Simon J. Gay. Bibliography on quantum programming languages. Available at <http://www.dcs.gla.ac.uk/~simon/quantum/>, 2005.
- [Gay06] Simon J. Gay. Quantum programming languages: Survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.
- [Hir01] Mika Hirvensalo. *Quantum Computing*. Springer-Verlag, Berlin, Heidelberg, 2001.
- [HW] Alexander S. Holevo and Reinhard F. Werner. *Introduction to Quantum Information*. Springer, Berlin, Germany, in preparation.
- [JM08] Richard Jozsa and Akimasa Miyake. Matchgates and classical simulation of quantum circuits. Available at <http://arXiv:0804.4050v2>, November 2008.
- [Joz08] Richard Jozsa. Embedding classical into quantum computation. Available at <http://arXiv:0812.4511v1>, December 2008.
- [Mer07] David N. Mermin. *Quantum Computer Science. An Introduction*. Cambridge University Press, Cambridge, 2007.
- [Mos08] Michele Mosca. Quantum algorithms. Available at <http://arXiv:0808.0369v1>, August 2008.
- [NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.
- [Rüd03] Roland Rüdiger. Quantenprogrammiersprachen. *Informatik-Spektrum*, 26(6):406–409, 2003. also in GI-online-lexicon.
- [Rüd07] Roland Rüdiger. Quantum programming languages: An introductory overview. *The Computer Journal*, 50(2):134–150, 2007.
- [Rüd09] Roland Rüdiger. Quantenprogrammierung. *Informatik-Spektrum*, 32(2):93–101, 2009.
- [Sel04a] Peter Selinger. A brief survey of quantum programming languages. In *Functional and Logic Programming: 7th International Symposium*, volume 2998 of *LNCS*, pages 1–6, Berlin, Germany, April 2004. Springer.
- [Sel04b] Peter Selinger, editor. *Proceedings of the 2nd International Workshop on Quantum Programming Languages*, volume TUCS General Publication No 33, Turku, Finland, July 12-13 2004. Turku Centre for Computer Science.
- [Sel05] Peter Selinger, editor. *Proceedings of the 3rd International Workshop on Quantum Programming Languages*, Chicago, USA, June 30 - July 1 2005. Electronic Notes in Theoretical Computer Science.
- [Sel06] Peter Selinger, editor. *Proceedings of the 4th International Workshop on Quantum Programming Languages*, Oxford, UK, July 17-19 2006. Electronic Notes in Theoretical Computer Science.
- [Sof08] Donald A. Sofge. A survey of quantum programming languages: History, methods, and tools. In *Proceedings of the 2nd International Conference on Quantum, Nano, and Micro Technologies (ICQNM 2008)*, pages 66–71. IEEE Computer Society, 2008.
- [SS04] Joachim Stolze and Dieter Suter. *Quantum Computing. A Short Course from Theory to Experiment*. Wiley VCH, Weinheim, 2004.
- [Unr06] Dominique Unruh. Quantum programming languages. *Informatik Forsch. Entw.*, 21:55–63, 2006.

—The End—

Thanks for your attention.