Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

2009

Quantum Inductive Learning and Quantum Logic Synthesis

Martin Lukac

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Lukac, Martin, "Quantum Inductive Learning and Quantum Logic Synthesis" (2009). *Dissertations and Theses*. Paper 2319.

<https://doi.org/10.15760/etd.2316>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

QUANTUM INDUCTIVE LEARNING AND QUANTUM LOGIC
SYNTHESIS

by
MARTIN LUKAC

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING.

Portland State University
2009

DISSERTATION APPROVAL

The abstract and dissertation of Martin Lukac for the Doctor of Philosophy in Electrical and Computer Engineering were presented January 9, 2009, and accepted by the dissertation committee and the doctoral program.

COMMITTEE APPROVALS:

Marek Perkowski, Chair

Garrison Greenwood

Xiaoyu Song

George Lendaris

Bart Massey

Steven Bleiler
Representative of the Office of Graduate
Studies

DOCTORAL PROGRAM APPROVAL:

Malgorzata Chrzanowska-Jeske, Director
Electrical and Computer Engineering Ph.D.
Program

ABSTRACT

An abstract of the dissertation of Martin Lukac for the Doctor of Philosophy in Electrical and Computer Engineering presented January 9, 2009.

Title: Quantum Inductive Learning and Quantum Logic Synthesis

Since Quantum Computer is almost realizable on large scale and Quantum Technology is one of the main solutions to the Moore Limit, Quantum Logic Synthesis (QLS) has become a required theory and tool for designing Quantum Logic Circuits. However, despite its growth, there is no any unified approach to QLS as Quantum Computing is still being discovered and novel applications are being identified.

The intent of this study is to experimentally explore principles of Quantum Logic Synthesis and its applications to Inductive Machine Learning.

Based on algorithmic approach, I first design a Genetic Algorithm for Quantum Logic Synthesis that is used to prove and verify the methods proposed in this work.

Based on results obtained from the evolutionary experimentation, I propose a fast, structure and cost based exhaustive search that is used for the design of

a novel, least expensive universal family of quantum gates.

The results from both the evolutionary and heuristic search are used to formulate an Inductive Learning Approach based on Quantum Logic Synthesis with the intended application being the humanoid behavioral robotics.

The presented approach illustrates a successful algorithmic approach, where the search algorithm was able to invent/discover novel quantum circuits as well as novel principles in Quantum Logic Synthesis.

To my parents and family.

ACKNOWLEDGMENTS

My sincere thanks to my advisor prof. Marek Perkowski and to each member of my dissertation committee for their guidance and help.

CONTENTS

<i>Acknowledgments</i>	ii
<i>List of Tables</i>	x
<i>List of Figures</i>	xii
<i>1 Introduction</i>	1
1.1 Dissertation Statement	3
1.2 Contributions	3
<i>2 Quantum Computing Basics and Concepts</i>	8
2.1 Why quantum computing?	8
2.2 Mathematical Preliminaries to Quantum Computing	10
2.2.1 Bra-Ket notation	12
2.2.2 Matrix Trace	12
2.3 Quantum Mechanics	13
2.3.1 Bohr Particle Model	13
2.3.2 Quantum Model of Elementary Particle	15

2.3.3	Schrödinger equation	20
2.3.4	Superposition of quantum states	21
2.4	From Quantum Mechanics to Quantum Logic	24
2.4.1	Simple Projective Measurement	29
2.4.2	Density Matrix and POVM	37
2.5	Quantum Logic, Quantum Gates and Quantum Logic Circuits	46
2.5.1	Single-qubit Quantum Gates	46
2.5.2	Multi-qubit and Controlled Quantum Gates	49
2.5.3	Quantum Circuits and Representation	52
2.5.4	Quantum Circuits and Sequential Logic	60
3	<i>Quantum Logic Synthesis and Algorithmic Search</i>	68
3.1	Introduction	68
3.2	Previous research on automated synthesis of quantum circuits	69
3.3	Function Classification for Logic Synthesis	71
3.4	Representation	72
3.5	Quantum-Based Synthesis: useful quantum circuits synthesis problems	75
3.5.1	Cost of quantum circuits	77
3.5.2	The Size of Quantum circuits	80

3.5.3	Quantum Logic Synthesis	83
3.5.4	NMR-based Quantum Logic Gates	86
3.6	Principles of Synthesis for NMR technology	88
3.6.1	V/T Gate - 'Level-generalization'	88
3.6.2	Insertion principle	91
3.6.3	The Divide and Conquer Principle	91
3.6.4	The Gate-collapsing principle	92
3.7	Examples of circuits obtained automatically for NMR technol- ogy using methods from sections 3.5 and 3.6	94
3.8	Chapter Conclusion	100
4	<i>Genetic Algorithm for Logic Synthesis</i>	101
4.1	Introduction	101
4.2	Genetic algorithm	105
4.2.1	Encoding/Representation	105
4.2.2	Initialization steps of GA	107
4.3	Evaluation of Synthesis Errors	111
4.3.1	Element Error Evaluation method (EE)	112
4.3.2	Measurement Evaluation method (ME)	113
4.3.3	Comparison of EE and ME	116

4.4	Fitness functions of the GA	122
4.4.1	Simple Fitness Functions	123
4.4.2	Cost Based Fitness Functions	124
4.5	The Selection Process	129
4.6	Crossover and Mutation	133
4.6.1	Mutation	135
4.6.2	Additional GA tuning strategies	138
4.7	Chapter Conclusion	142
5	<i>Evolutionary Search for Logic synthesis</i>	144
5.1	Evolutionary Search	146
5.2	Experimental setup of GA	147
5.2.1	Input Sets of Quantum Primitives	147
5.3	Discussion of the Results of the Evolutionary Quantum Logic Synthesis using ME evaluation	150
5.3.1	Toffoli Gate	150
5.3.2	Synthesis of Fredkin Gate	159
5.3.3	Synthesis of Majority Gate	166
5.4	Discussion of the Results of the Evolutionary Quantum Logic Synthesis using EE evaluation	173

5.4.1	Toffoli Gate	173
5.4.2	Fredkin Gate	176
5.4.3	Entanglement Circuit	177
5.5	Discussion of the results of the Evolutionary Synthesis	180
5.5.1	Results Comparison	180
5.5.2	Encountered problems during the Evolutionary QLS . .	181
5.6	Conclusion of the Evolutionary QLS	186
6	<i>Structure Based Search for Universal Quantum Circuits</i>	188
6.1	Introduction	188
6.2	The EX algorithm	191
6.3	Heuristic Search for Lowest Cost Function Class using the EX algorithm	195
6.3.1	New Quantum Logic Family	197
6.4	Discussion and Conclusion to the Structure Based Exhaustive search	201
7	<i>Learning Quantum Behaviors</i>	206
7.1	Quantum Behaviors	208
7.2	The concept of learning robotic behaviors from examples. . . .	218

7.2.1	Symbolic Quantum Synthesis of Single Output Quantum Circuits	233
7.3	Experiments and Results for learning Quantum Behaviors . .	239
7.3.1	Symbolic synthesis - Single output functions	239
7.4	Discussion on learning Quantum Benchmarks	242
7.5	Measurement Synthesis	246
7.5.1	Conclusions on the Measurement Dependent Quantum Logic Synthesis	265
8	<i>Dissertation conclusion and contributions</i>	268
	<i>References</i>	271
	<i>Appendices</i>	287
A	<i>Glossary</i>	288
B	<i>Models of Quantum Computing</i>	290
B.0.2	Quantum Cellular Automata	290
B.0.3	Quantum Finite Automata and Quantum State Machines	301
B.0.4	Quantum Turing machine	316
B.0.5	Quantum Robots	317

<i>C Cynthea, an interactive robotic system</i>	<i>321</i>
C.1 Definitions	321
C.2 Cynthea - the social interactive robotic framework	322
C.3 Learning and Adaptation	323
C.4 Sensors	327
C.4.1 Video-Sensor	328
C.4.2 Audio-Sensor	331
C.4.3 Voltage/Power-Sensor	333
C.5 Actuators	334
C.5.1 Servo-Control	336
C.5.2 Speech-Control	337
C.5.3 Network Control	338
C.5.4 Speech and Dialogue generation	339
C.6 Control and Scripting	341
C.6.1 CRL - The language	344
C.6.2 Scripting	355
C.7 Artificial Emotions	357

LIST OF TABLES

2.1	a). - K-map , b) - LUT	52
2.2	K-map of an incompletely specified 3×1 reversible quantum function before measurement	56
2.3	K-map of an incompletely specified 3×3 reversible quantum function before measurement with don't cares within single minterms	58
2.4	K-map of the Feynman gate	64
3.1	Cost models	78
4.1	Structure of a Genetic Algorithm	105
4.2	Example of the Majority gate encoding for the GA. Observe that each input minterm with qubit $ q_3\rangle = 1\rangle$ is a don't care. .	115
5.1	Fixed parameters during search for Toffoli gate	150
5.2	Example of a 3×3 quantum probabilistic and a 3×1 determin- istic functions generated by the circuit from Figure 5.4c. . . .	159
5.3	Fixed parameters during search for Fredkin gate	159
5.4	Comparison of various fitness functions for all illustrated bench- mark functions	181
6.1	Pseudo Code of EX Algorithm	192

6.2	The Peres gate Truth Table	196
6.3	The family of Peres gates	198
7.1	Braitenberg vehicle behaviors: (a) <i>afraid of light</i> , (b) <i>attracted to light</i>	209
7.2	A K-map of a completely specified reversible function	224
7.3	Incompletely specified reversible function f	227
7.4	Single output Boolean functions R to be synthesized (learned)	233
7.5	Example of analysis on circuit from Figure 7.11	236
7.6	Example of analysis on circuit from Figure 7.12	238
7.7	GA benchmark functions	239

LIST OF FIGURES

2.1	Bohr model of the atom (nucleus, orbiting electrons). Shown are light colors respective to the electron orbit transitions. . .	16
2.2	The measured number of electrons on the detector screen with top or the bottom slit open (thin lines). The expected probability when both slits are open (thick line).	17
2.3	Results of measurement of particles position when both slits of the screen are open.	18
2.4	Schematic representation of the two-slit experiment. Left is the emitter and on the right is the detector (film). In the middle is the barrier with two holes.	18
2.5	Circuit representing the $W \otimes H$ operation	27
2.6	EPR producing circuit	31
2.7	Schematic representation of Controlled-U gates: a) general structure of single-qubit controlled U gate (control qubit a, target qubit, b) two-qubit controlled single qubit operation, c) single-qubit controlled two-qubit target quantum gate, d) Feynman (CNOT), e) Toffoli (CCNOT), f) Fredkin. a, b, c are input qubits and a', b' and c' are respective outputs.	49

2.8	Example of representation of a quantum circuit using a quantum K-map. X is the Pauli X rotation or an Inverter.	54
2.9	Example of a quantum circuit (a) having a non-permutative unitary matrix (using a quantum K-map (b)). The V is the $\sqrt{X}$ gate.	55
2.10	Schematic representation of detecting the quantum states V_0 and V_1 using the density matrices D_0 and D_1	59
2.11	Space-time representation of a classical logic circuit. Time flows from left to right along the X axis (Y axis represents space dimension), unless a vertical interconnection is made in which case time also flows along the Y direction, following the direction of arrows.	61
2.12	Space-time representation of a quantum logic circuit without measurement. Time flows from left to right along the X axis while space is represented by the Y axis. Remark that the physical size of this circuit is equal only to its width.	62
2.13	(a) - The state diagram for a FSM using a single Feynman Gate, (b) - schematic representation of a FSM built according to the protocol described above.	65
3.1	Schema representing the tree of relations between synthesis approaches for completely specified and incompletely specified quantum/reversible functions as used in this dissertation.	71

3.2	Toffoli gate realized using 2×2 controlled quantum gates. When used as a quantum robot controller, signals a, b and e can come from touch, sound or other sensors and outputs P, Q and R through measurement units go to motors or other actuators. .	75
3.3	Structure of the Peres gate built from 2×2 quantum primitives	79
3.4	Structure and minimization stages to reduce the cost of the Fredkin gate. The top diagram shows the circuit non minimized. The middle diagram represents the circuit with permuted gates to allow forming larger blocks. The bottom diagram shows the minimized circuit for which the total quantum cost is calculated [LPG ⁺ 04].	80
3.5	Example of Blocks of a Quantum Circuit.	81
3.6	Example of Segmentation of a Quantum Circuit. In this case, the circuit is built as a serial composition of three parallel segments a, b and c.	82
3.7	Example of Segmentation of a Quantum Circuit. In this case, the circuit is built from three parallel segments a, b, c and d. .	82
3.8	Structure of the Toffoli gate	86
3.9	Single pulse Logic gate - NOT	87
3.10	Two pulses Logic gate - Hadamard	87
3.11	Detailed Realization of Feynman gate with five EM pulses. . .	88

3.12 Five pulses Logic gate - Controlled-V	88
3.13 SWAP gate broken into two T gates	90
3.14 Fredkin gate broken into two Controlled-T (<i>CT</i>) gates	90
3.15 The level-generalization of Peres using <i>CT</i> . From a three-qubit circuit with (<i>CV</i> and <i>CNOT</i> gates) at the left, the four qubit circuit (using <i>CNOT</i> and <i>CS</i>) is related as the "Level-generalization".	91
3.16 Peres, Toffoli and Fredkin gates illustrating simple search technique (PTF principle).	92
3.17 The "divide and conquer" circuit synthesis method	92
3.18 The gate-collapsing principle: quantum gates located on the same qubits and having no other gates in between can be <i>collapsed</i> into a single one.	93
3.19 The gate-collapsing principle: quantum gates can also be collapsed if they can be moved and become neighbors without altering the controls of other quantum gates.	94
3.20 Toffoli circuit from 2×2 quantum primitives	96
3.21 Toffoli-based Fredkin circuit from Feynman gates and a macro-Toffoli gate	96

3.22	Stages of the minimization of the Miller Gate. Observe that by using the Peres macro and the collapsing principle the overall cost of the gate is reduced. The important fact is that the collapsed blocks C_V and $C_{V\uparrow}$ can be much less expensive in some technologies than in others.	98
3.23	Example of comparison of synthesis using Toffoli and Peres primitives.	99
4.1	Example of an individual in a GA solving the TSP problem. .	103
4.2	Example of cross-over operation for the TSP problem.	104
4.3	Transformation of a QC from the chromosome (on the top) encoded string, to a final quantum circuit notation representation of this QC (on the right). Here SW is a Swap gate, H is a Hadamard gate and I is a Identity. In the middle there is one $CCNOT$ (Toffoli) gate.	106
4.4	Majority gate function: (a) the K-map, and (b) the schematic circuit representation.	116
4.5	Schematic representation of the four available fitness functions. The solid lines represent respectively the fitness functions f_1 and f_2 and the dashed lines represent the fitness functions f_3 and f_4	125
4.6	The fitness normalization represented on a line (a roulette wheel). Each individual has a segment allocated proportional to its fitness	131

4.7	The Roulette Wheel Selection - a random number generator picks the individuals one by one located on the equi-distant points over the individuals fitness-sum representation. Here, $p \approx 0.45$ selects individual c_4	131
4.8	For N given individuals to select, pick a random number, and from its location pick N-1 equidistant points on the fitness landscape. In this case the selected individuals are c_2, c_4, c_7 and c_{10}	132
4.9	single-point crossover	134
4.10	Two point crossover: in both parents, two random points are selected and the segments between such points from each parent are exchanged.	135
4.11	<i>Flip</i> mutation a) bitwise <i>Flip</i> mutation b) and <i>Swap</i> mutation c)	136
4.12	Replacement of a segment - the first mutation method	137
4.13	Erasure of a segment - the second mutation method	137
4.14	Adding of a segment - third mutation method	138
4.15	Example of the repair process. The repair mechanism is able to break down individual segments of gates and reconstruct them so as the circuit remains structurally valid.	142

5.1	Results for Toffoli gate using the single-qubit measurement for two different settings of t_{min} and t_{max} . Observe the overall convergence of the evolutionary search in (a) and compare to relative non-convergence in (b) due to a over-estimated parameters t_{min} and t_{max}	152
5.2	Results for Toffoli gate using the multiple-qubit measurement for two different settings of t_{min} and t_{max}	154
5.3	Example of successful synthesis of the Toffoli gate using a partial input gate-set S_p and a single-qubit ME method.	157
5.4	Circuits discovered by the GA as a single output Toffoli Gate using the single-qubit ME evaluation. (Unitary matrix representing it is given in eq. 5.1)	158
5.5	Example of realization (with minimization) Fredkin gate using single qubit ME model.	160
5.6	Example of realization (with minimization) of Fredkin gate using two-qubit ME model and the S_f input-gate set. Observe that all presented gates contain the well known realization of Fredkin ($[NOTC]$ $[TOFFOLI]$ $[NOTC]$) on the qubits b and c, but with additional gates changing the value on the third, unmeasured qubit a.	161

5.7	Comparison of results for Fredkin gate using the full input gate-set S_f for two different settings of t_{min} and t_{max} . Similarly to the case of Toffoli gate, the overestimation of the size parameters leads to poorer results.	163
5.8	Comparison of results for Fredkin gate using the partial input gate-set S_p and for two different settings of t_{max} and t_{min} . (a) - the value of parameters t_{max} and t_{min} is too small for the GA to find a solution and thus the statistical average is stuck (shown as a straight line on the top) in a locally best solution. (b) - an increase in size allows the GA to explore larger circuits and also to find solutions, despite the fact that overall convergence is not visible.	164
5.9	Example of successful synthesis of the Fredkin gate using a full input gate-set S_f and a two-qubit ME method.	165
5.10	Example of synthesized Majority circuit with the consequent minimization and final result (using ME evaluation).	166
5.11	Comparison of three different setting for minimal (t_{min}) and maximal (t_{max}) number of circuit segments during the search for Majority gate using the single-qubit ME.	169

-
- 5.12 Example Majority circuit found by the GA. The Figure also shows steps of the minimization process. Observe that the aggregate gate C_0 is created by moving the $CV^\dagger$ gate near the CNOT gate (left-to-right arrow in the middle circuit). 171
- 5.13 Example of successful synthesis of the Majority gate using a partial input gate-set S_p and a single-qubit ME method. . . . 172
- 5.14 Various realizations of Toffoli gate obtained from GA before their optimizing transformations were applied. (a) - three CNOT gates (the dashed frame 1) are collapsed into a single one, (b) - the two Z gates on the topmost wire can be removed as they do not change the observable of the circuit (after measurement), (c) - the frame 1 is collapsed to a single two-qubit gate, frame 2 is removed as $[CNOT] \times [CNOT] = [I]$ and frame 3 is collapsed to a single CNOT gate, (d) - is minimized to the well known minimal Toffoli gate by removing the two NOT gates and transforming two CV gates into a single CNOT. 173
- 5.15 Results of synthesis of Fredkin gate. *a)* is result of application of fitness from eq. 4.15 and a complete set and *b)* the result of fitness from 4.17 and a complete set. Figure *c)* and *d)* represents results obtained using the same settings but without using the SWAP gate in the input gate-set. 176

5.16 Results of Synthesis and Minimization of the Entanglement circuit for 2 qubits.	178
5.17 Results of Synthesis and Minimization of the Entanglement circuit for 3 qubits.	179
5.18 Results of Synthesis and Minimization of the Entanglement circuit for 4 qubits.	180
5.19 Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.	182
5.20 Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.	183
5.21 A comparative chart between two different fitness functions during the same GA run.	184
5.22 Inversely constructed CNOT generates CE	187

-
- 6.1 Figure (a) illustrates an unsuccessful GA run with the final solution having fitness value $\sim 80\%$ and a cost close to the minimal known cost. In such a case one of the solutions generated during this run can be used as a template for the exhaustive search. This is represented in (b) where each line is the path of a GA run through the problem landscape, with a deeper exhaustive search at the end (represented by a circle). The axes in (b) are representing a parametric space defined by the possible quantum circuits that the GA can synthesize during the evolutionary search. 190
- 6.2 Example of segment encoding for exhaustive search. Left bottom is an example of encoding segments to gates while right bottom shows an example where each gate becomes a segment. 191
- 6.3 Random permutative circuit 194
- 6.4 All possibilities of placing a Feynman gate on three qubits . . 196
- 6.5 (a) Circuit representing function $f(a, b, c) = [0, 1, 2, 3, 7, 6, 4, 5]$,
 (b) Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 6, 7, 4, 5]$. 197
- 6.6 (a) Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 7, 6, 5, 4]$, (b)
 Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 7, 6, 4, 5]$. . 198
- 6.7 Circuit realizing the reversible function $f = [0, 1, 3, 2, 7, 6, 4, 5]$.
 Each CNOT gate was broken down to two *CV* gates. 200

6.8	Circuit representing function $[0, 1, 3, 2, 6, 7, 4, 5]$, Circuit representing function $[0, 1, 3, 2, 7, 6, 5, 4]$	200
6.9	Circuit representing function $[0, 1, 2, 3, 7, 6, 4, 5]$, Circuit representing function $[0, 1, 2, 3, 6, 7, 5, 4]$	201
6.10	Permutative circuits obtained using only CV and CNOT gates. a) Observe that only gate CV is used, thus a new principle of creating permutative gate was discovered by the EX algorithm, b) by simply adding a Feynman gate, the circuit realizes now both a multiplication and addition.	203
6.11	Trivial alteration of the circuit from Figure 6.10 (changing a single CV gate to $CV^\dagger$ and adding an inverter on qubit a. . .	203
6.12	Inverse non trivial alteration of the function $[0, 1, 3, 2, 7, 6, 5, 4]$ (generating a new function on the circuit outputs)	204
7.1	Circuits representing behaviors (a) <i>afraid of light</i> and <i>attracted to light</i> as shown in Table 7.1.	209
7.2	Circuit representing a robotic controller (with multiple robotic behaviors (labeled B_k)) with two input sensors and two motor control outputs. Each B_i is unitary matrix of certain elementary behavior.	210
7.3	Schematic diagram of circuit showing two behaviors controlled by a control bit. This bit can come from another input, the user of the robot or some higher-level quantum or classical controller.	212

-
- 7.4 Schematic diagram of circuit showing two behaviors controlled by a control input qubit and with dedicated qubits for the outputs. 215
- 7.5 a) - The state diagram of the machine represented by the state transition function U_0 , b) - the schematic diagram of the circuit showing the initialization, the Unitary operator and the measurement on the output qubit. 216
- 7.6 Schematic diagram of the circuit showing the initialization, the Unitary operator and the measurement gates on both the output and the state qubits. 218
- 7.7 a) Schematic representation of Concept Learning with C representing the definition of the problem, C' representing the concept to learn. δ represents the error (difference between the original concept C and the target concept C'). The hexagon in the bottom left corner represents the operation of transforming the target concept C' by each query to the Examples (adding or removing minterms). The oval shaped object with $(1||0)$ inside in the upper right corner represents the result of querying the Oracle with an generated example vector of C' resulting either in 1: the Oracle confirms that the given vector is an element of C , or in 0: the vector does not represent the concept class C .
b) - representation of the Logic Synthesis process. 223

7.8	Example of a completely specified 3×3 reversible function realized as a quantum circuit using quantum primitives CV , CV [†] and CNOT (CNOT).	226
7.9	Three modes of learning based on the properties of quantum systems. On the left the desired function is represented using cares and don't cares. The top Karnaugh Map represents the result of the mapping being the deterministic learning. The bottom map represents probabilistic and the map on the right represents the learned function using Measurement dependent output.	230
7.10	Schematic representing the possible learning methods in Quantum Computing. As defined in this dissertation, using QLS, the learning can be done for the data specified either before or after the measurement. Learning can be also done on a single or multiple qubits.	231
7.11	Example of a solution to the learning problem for the incomplete function from Table 7.4. This circuit has a deterministic behavior.	236
7.12	Example of learning from a set of example pairs corresponding to one output of a Toffoli gate ($ab \oplus c$). Observe that when the control outputs are not forced to be restored, the overall cost of the gate is smaller.	237
7.13	Benchmark function $f_4(a, b, c)$	240

7.14	Benchmark function $f_5(a, b, c)$	241
7.15	Function 3×1 Majority.	241
7.16	Explanation of two models of quantum learning: Learning 1 represents the method that approximates complex coefficients as represented by the quantum states (vectors of Hilbert space) in the quantum circuit. Learning 2 represents the learning of binary vectors after the measurement (observable). This second method evaluates the learning result (the circuit) only on its observable behavior.	245
7.17	The schematic of a quantum controller used in a classical robot. All operations are synchronized by one or multiple clocks assuring the correct outputs are generated for the appropriate inputs. Blocks S and O represent the classical sensors and actuators wires, I and M represent the initialization and the measurement operations of the quantum controller and U is the actual operation (unitary transformation) applied on the quantum circuit.	247
7.18	The application of the NOT gate to the entangled system creates rotations of pairs of vertices (minterms, K-map cells) within the hypercube.	249

- 7.19 By pre-storing in additional non-quantum circuit the possible outcomes of a single qubit $|d\rangle$, the desired function can be realized using the entanglement operation. With measurement outcome $m = 0$ on qubit $|d\rangle$ the state is $|abcd\rangle = |0110\rangle$ and with measurement outcome $m = 1$ the state is $|1001\rangle$. Thus with single measurement 0 the control state 0110 is given to the actuators of the robot. 250
- 7.20 Schema representing the setting of measurement dependent computation for robotic control. (a) - a quantum circuit implementing the entanglement, (b) - the LUT storing the control signals for the robot and allowing to associate single measurement output with appropriate multi-qubit control signal and (c) the classical wires for the actuator control. The binary signals of the actuator wires are a function of inputs, of the measurement in qubit $|c\rangle$ and of the logic function stored in ROM. Observe, that the addressing mechanism is using classical inputs from the sensors; the quantum circuit is initialized only after the address has been assigned in order to avoid entanglement between the quantum circuit and the addressing mechanism. 251

-
- 7.21 Entangled circuit representing the mechanism of selecting the desired function. The operator M_n is the conditional operator that allows to select correct expected measured values. The choice of the operator M_n is decided by the ROM at right. In this case, the addressing mechanism is using measured quantum inputs from the sensors; the quantum circuit is initialized only after the address has been assigned in order to avoid entanglement between the quantum circuit and the addressing mechanism. 252
- 7.22 The Measurement-dependent Synthesis: inputs are measured and the observable values determine which measurement is to be applied to the target qubits. This mechanism is also used to select the pre-stored output states in the MDQC 253
- 7.23 Schematic diagram of a quantum robot in a real environment. The Controls qubit are not measured and in combination with the discrete Sensor inputs they can generate a mixture of behaviors on the output as well as many quantum internal states. 258

-
- 7.24 The circuit representing function f (eq. 7.19) and represented as in eq. 7.18. Observe that in this case the measurement is controlled by quantum signals that are copied using a set of CNOT gates and measured afterwards. Unlike in Figure 7.21 this setup allows more freedom in the sense that the addressing mechanism can now select a cell from the ROM with a probability rather than deterministically. 261
- 7.25 Example of implementation of the Toffoli gate using measurement-dependent reversible implementation for a single bit correct output. In this case, observe that the single measured qubit represents the Toffoli gate, but the overall output (three bits) is now a relation. The selection of the pre-stored values in the output LUT uses the selection method from Figure 7.22, but here there are three measurements in addresses of ROM and only one qubit is conditionally measured ($|c\rangle$). Observe that for clarity the output ROM contains the whole output states, but the real content of the ROM is only the bold value that is used to select the appropriate measurement operator. Also, the output signals (A) are connected to the qubits of the circuit with dotted lines representing the fact that the output is selected only conditionally (based on the result of the measurement). 264

- 7.26 Another possible realization of the circuit realizing the measurement-dependent Toffoli function. (a) the circuit uses partial input state and the output of the measurement to select the correct output state. In this case the complete output states are stored in classical memory and the output of the circuit is used only for retrieving the content from the ROM.(b) a MUX representing the selection of the output based on input and single qubit measurement. Remark, this is different from the previous approach where the measurement is used to decide which of the possible outputs is the correct one. Similarly to Figure 7.25 the output signals are conditionally (measurement-outcome dependent) assigned to output wires (A). (Figure 7.21) 265

1 INTRODUCTION

Despite the fact that the physical realization of quantum computers is still in experimental state (current technologies such as NMR, Ion Trap, QED Cavity or Josephson junction allow quantum logic circuits only on a relatively small scale (12 qubits)), various approaches to Quantum Logic Synthesis (QLS) based on quantum logic have already been proposed. The motivation for QLS is increasing as with every year the limit of Moore's law is being slowly reached. The potential realization of quantum computation is motivated by reversibility and power-saving; it was Landauer [Lan61] who has shown that with the constant decrease in technology-related energy consumption and increase of the scale of the electronic components, there is an energy barrier under which it is not possible to minimize energy loss without considering the information loss [Lan61]. Information is lost when energy is lost and information is not lost at all in a reversible (quantum) circuit. With the current progress in technology and in VLSI (Very Large Scale of Integration), quantum computation is an ideal candidate to realize future power-saving permutative or unitary quantum logic circuits.

Quantum computation is reversible and, in the ideal case, almost lossless particle based computation can be built in the future. Because for some classes of problems quantum logic provides up to an exponential speed-up in com-

putation [Sho94, DJ92] and the goal of QLS is the building of permutative functions (boolean reversible functions realized in quantum logic), the synthesis of quantum circuits is an interesting and open topic with potentially great future.

Machine learning is also a well-established research area with many applications in various engineering areas. 'Concept Learning' is a sub-class of machine learning and represents a methodology to design functions, concepts and machines from examples [Val84]. This area is also known as Probably Approximately Correct (PAC) learning. In this approach the problem is given as a set of samples representing the knowledge about the target concept, and the goal is to learn a concept minimizing the error between the learned and the desired concept in polynomial time [Val84]. This formulation is very appropriate for machine learning; in general the problem specifications contain a large amount of unknown information about the environment or states, generally available as a set of examples. Various work has been done in exploring Logic Synthesis (LS) tools and methods for concept finding, problem minimization or problem decomposition [Gry00, Zup97, Lan02]. In classical computing, the link between machine learning and LS has already been explored. In Quantum Computing, however, the amount of work in this area is much more restricted.

1.1 Dissertation Statement

The problems addressed in this dissertation are: the automated design of quantum logic circuits, the search for quantum circuits and the use of QLS for learning robotic behaviors (synthesize quantum circuits) from examples. Each problem is studied using methodologies described below.

- *Synthesis of Quantum Circuits* - Use genetic algorithm to minimize, synthesize and optimize quantum circuits.
- *Generalization of functions to cost-equivalent classes of functions by synthesis of quantum circuits*- Search (using heuristics) quantum logic space for function generalization.
- *Quantum machine learning based on QLS*- Develop effective definition of inductive quantum machine learning based on quantum circuit synthesis.

1.2 Contributions

- *Synthesis of Quantum Circuits* .

We show that small quantum circuits and automata can be constructed automatically from examples. A set of examples represents a set of Input-Output pairs specifying the behavior of a certain system, for instance a discrete function. In this approach, we consider functions where the set of examples describes only partially the function as well as those where a set of examples describes completely the function to be synthesized.

The process of synthesis, re-synthesis and optimization is performed by a Genetic Algorithm that is specially constructed for this purpose. This GA has a particular encoding of quantum circuits that allows it to minimize and represent quantum circuits simply and efficiently. At the time of its implementation, this particular GA was one of the first Genetic Algorithms for QLS[LPG⁺03, LPG⁺04]. The importance of this study is in the fact that we experimentally describe the relation between the cost of quantum circuits and their function and thus allow for more precise evaluation functions for QLS. A well defined relation between the cost of quantum circuits and its function allows for the development of efficient algorithms for QLS.

- *Generalization of functions to cost-equivalent classes of functions by synthesis of quantum circuits.*

We describe an approach to the generalization of the class of cost equivalent quantum functions from the cheapest universal quantum gate, Peres gate [Per00]. In this approach, we search the problem space of quantum logic circuits that is limited to only the component gates from which the Peres gate is built. Moreover, the circuits are generated for a very small number (maximum 7) of components in order to remain computationally tractable. We demonstrate, that despite the extremely large problem space with many local maxima, in a well defined QLS framework it is possible to use exhaustive search for function synthesis and generalization. By introducing design constraints and heuristics, we show that

a set of gates can be developed to reinvent the minimal universal Peres gate and create the next family of universal gates with the same cost but with various functionalities. This family of gates is less expensive than the well known Toffoli or Fredkin and it can be used for more efficient circuit minimization and synthesis. The importance of this approach to QLS is that it allows one to create relatively easily certain sets of universal quantum gates with minimal costs, and thus creating new libraries of quantum gates for future synthesis problems. Moreover the logic synthesis that uses this new family of gates generates cheaper circuits for some sets of functions and thus has a real life applications in future circuit design.

- *Quantum machine learning based on QLS.*

The proposed machine learning method is based on the heuristics discovered from the experimentation with the Genetic algorithm. Two methods of learning are presented: first a direct application of inductive bias in QLS is used to synthesize arbitrary quantum logic function, second, a method that uses the measurement as a computational element is presented. In particular I define a quantum inductive learning approach that builds quantum robotic behaviors that potentially are more powerful than permutative or probabilistic unitary behaviors. By permutative behaviors we mean behaviors of circuits specified by permutative matrices and by unitary behaviors we mean circuits described by unitary matrices. The introduced machine learning methods are novel because

they are QLS-based, and also because the measurement operation is used as a computational element. The utility of this approach is that the inductive bias of a quantum logic synthesizer is different than inductive bias of a classical synthesizer (which builds binary reversible and non-reversible circuits) and thus the approach defines a new type of learning - quantum learning in the proper sense. This learning model includes therefore classical and probabilistic learning as well as unitary behavior learning as a special case. The importance of this research lies in the fact that I demonstrate that quantum computing can be used in another way for learning and computation by using not only well known components as the unitary transformation and measurement but also by combining quantum computation and elements from classical (non-quantum) computation into a novel approach. The proposed technique for learning can be used to design hybrid (both classical and quantum) algorithms for learning and problem solving that would not only use features of quantum computing but also elements of classical computing. In this case the application is in humanoid robotics.

This dissertation is organized as follows. In chapter 2 Quantum Computing and related concepts are introduced. Chapter 3 presents Quantum Logic Synthesis as well as the methods used in this dissertation. Chapter 4 describes the Genetic Algorithm designed for the Quantum Logic Synthesis and used for the experiments. Chapter 5 shows the results from the QLS of universal quantum gates and other benchmarks. Chapter 6 presents the exten-

sion of the GA called the GAEX (and the stand-alone version called EX) and the experimentation resulting in the invention of a family of universal quantum gates. Chapter 7 explains the QLS based approach to machine learning. Finally a dissertation conclusion is in chapter 8. Additionally, a glossary is in Appendix A, Appendix B presents common computing models in quantum computing and Appendix C describes the robot Cynthea that was built as illustration of my work and methodology.

2 QUANTUM COMPUTING BASICS AND CONCEPTS

2.1 Why quantum computing?

Quantum Mechanics (QM) describes the behavior and properties of elementary particles (EP) such as electrons or photons at the atomic and subatomic levels. Formulated in the first half of the 20th century mainly by Schrödinger, Bohr, Heisenberg and Dirac, it was only in the 80's of the same century that Feynman proposed the first physical realization of a Quantum Computer [Fey85]. In parallel to Feynman, Benioff [Ben82] also was one of the first researchers to formulate the principles of quantum computing and Deutsch proposed the first Quantum Algorithm [Deu85]. The reason these concepts are becoming popular in the logic design community is mainly due to the Moore's law [Moo65]; that is: the number of transistors in a chip doubles every 18 months and the size of gates is constantly shrinking. Consequently, problems such as heat dissipation and information loss are becoming very important for current and future technologies. Improving the scale of transistors ultimately leads to a technology working on the level of elementary particles (EP) such as single electrons or photons. Since Moore's paper, the progress led to the current $0.1\ \mu\text{m}$ (10^{-7}m) circuit technology which (considering the size of an atom, approximately 10^{-10}m) is relatively close to the atomic size. Consequently the exploration of QM and its related Quantum Computing becomes very impor-

tant to the development of logic design of future devices and in consequence to the development of quantum Computer Aided Design (CAD) and quantum logic synthesis methodologies and theories. Because of their superior performance and specific problem-related attributes, quantum computers will be predominantly used in AI and robotics, and similarly to classical computers they will ultimately enter every area of technology and day-to-day life.

Despite being based on paradoxical principles, QM has found applications in almost all fields of scientific research and technology. Yet the most important theoretical and in the future also practical innovations were done in the field of Quantum computing, quantum information, and quantum circuits design [BBC⁺95, SD96].

Although only theoretical concepts of implementation of complete quantum computer architectures have been proposed [BBC⁺95, Fey85, Ben82, Deu85], continuous progresses in technology might allow the construction of Quantum Computers in the near future, perhaps in the interval of 10 to 50 years. Recent implementation of small quantum logic operations with trapped atoms or ions [BBC⁺95, NC00, CZ95, DKK03, PW02] are an indication that this time-frame of close future can be potentially reduced to only a few years before the first fully quantum computer is constructed.

This chapter presents the basic concepts of quantum computing as well as the transition from quantum physics to quantum computing. We also introduce quantum computing models, necessary to understand our concepts of

quantum logic, quantum computing and quantum logic synthesis. The first section introduces some mathematical notions required for the understanding of quantum computing. The second section presents a historical overview of the quantum mechanical theory and the third section presents the transition from quantum mechanics to quantum logic circuits. The fourth section describes features and computation in the quantum computing paradigm and explains the quantum effects. The fifth section describes quantum logic gates and the principles of building quantum circuits. Finally, last section describes some computational models used in quantum computing in more details.

2.2 Mathematical Preliminaries to Quantum Computing

According to [Dir84] every physical system is associated with a separate Hilbert space H . An H space is an inner product vector space where the unit-vectors are the possible states of the system. An inner product for a vector space is defined by the following formula:

$$\langle x, y \rangle = \sum_k x_k^* y_k \quad (2.1)$$

where x and y are two vectors defined on H and x^* denotes a complex conjugate of x . As will be shown later, the eq. 2.1 can be written in the Dirac notation as: $\langle x|y \rangle$. For quantum computation it is important to introduce an orthonormal basis on H , in particular considering the $\frac{1}{2}$ -spin quantum system that is described by two orthonormal basis states. An orthonormal set of vec-

tors M in H is such that every element of M is a unit vector (length one) and any two distinct elements are orthogonal.

Example 2.2.0.1 Orthonormal basis set

An orthonormal basis set can be defined such as: $\{(1, 0, 0)^T, (0, 1, 0)^T, (0, 0, 1)^T\}$.

In this space, a linear operator A represented by a matrix A transforms an input vector $\mathbf{v}$ to an output vector $\mathbf{w}$ such as $w = Av$.

The multiplication of matrix A by vector $\mathbf{v}$ is defined by the following equation:

$$w[r] = \sum_c A[r, c] * v[c] \quad (2.2)$$

where r is the index of rows and c is the index of columns of the matrix. Such an operator is bounded in the sense that it maps bounded sets to bounded sets.

From equation (2.2), it follows that A is a projection, then $\langle Av, v \rangle = \|Av\|^2$ is called the *l2-norm* and measures the distance between the original vector $\mathbf{v}$ and the resulting vector $A\mathbf{v}$. The A operator is called Hermitian if its hermitian conjugate $A^\dagger = A$ (conjugate transpose) and a further extension of this property yields a *unitary* operator A . Such a unitary operator is invertible and its inverse is given by its conjugate transpose $A^\dagger$ (also called Hermitian adjoint).

Because all measurements of quantum events are of a probabilistic nature, the outputs are binary events (vectors) with probabilities in interval $\{0, 1\}$ and

the range of a projection is closed. The l2-norm of a projection of the vector $\mathbf{v}$ by A can be interpreted as a probability that a measurement will observe the system in the state represented by $A\mathbf{v}$. The interested reader can find more information about the Hilbert space and quantum-probabilistic systems in [WG98, HSY⁺04, YSHP05].

2.2.1 Bra-Ket notation

One of the notations used in Quantum Computing is the so called bra-ket notation introduced by Dirac [Dir84]. It is used to represent the operators and vectors; each expression has two parts, a bra and a ket. Each vector in the H space is a ket $|\Phi\rangle$ and each continuous linear function from Hilbert space H to complex space C is a bra $\langle\Psi|$. The application of bra to ket results in the bra-ket notation $\langle| \rangle$. In a orthonormal Hilbert space this results in: $\langle\psi_m|\psi_n\rangle = 1$, for $n = m$ being an inner product and $|\psi_m\rangle\langle\psi_n|$ being the outer product (or also called Kronecker product).

2.2.2 Matrix Trace

A trace of a matrix is defined as $\text{tra}(U) = \sum_i D_{ii}$ and as it will be seen it is used for the measurement operation in quantum computing. In particular it is required when dealing with ensemble systems [CFH97, NC00] and estimating their state. Such systems are represented by density matrices of the form:

$$\rho = \sum_i^{2^n} \alpha_i |\psi_i\rangle \langle \psi_i| \alpha_i^* = \sum_i^{2^n} p_i |\psi_i\rangle \langle \psi_i| \quad (2.3)$$

with $\sum_i^{2^n} p_i = 1$, α being the complex coefficient such that $|\alpha_i|^2 = p_i$.

The trace operator represents the possible observable states of a quantum system. This is because any quantum state $|\phi\rangle$ when observed collapses according to the applied measurement resulting in $\alpha|\phi\rangle \rightarrow p|\phi\rangle\langle\phi|$, with p being the probability of observing the state $|\phi\rangle$ from the set of all possible output states. Thus representing the overall state of a quantum system can be represented as $\sum_{i=0}^{2^n} p_i |i\rangle \langle i|$ with p_i being the probability of observing the state $|i\rangle$.

2.3 Quantum Mechanics

2.3.1 Bohr Particle Model

The term "quantum" describes the fact that the EP's can be found only in distinct energetic states and while moving from one state to another a quantified amount of energy is either emitted or absorbed. A closer look at the Bohr model of the atom will explain these notions even more. The example we are using here is based on the simplest of all atoms, the Hydrogen (H). As all atoms, the Hydrogen atom (H) is composed of a nucleus and electrons orbiting around it, but H has only one electron (e). The electron can be only on orbits of certain allowed radii. When e is on the orbit that is closest to the

nucleus then the atom is in the "ground state".

The electron can change orbits; going from a lower orbit to a higher one requires energy absorption and leaving an orbit for a lower one is characterized by emitting a quantum of energy from the electron. The energy levels that the electron can visit are characterized by the following equation:

$$E_n = (R_h) \left(\frac{1}{n^2} \right) \quad (2.4)$$

where R_h is the so-called *Rydberg constant* ($2.18 \times 10^{-18} J$) and n is the *principle quantum number* corresponding to different allowed orbits of the electron. The difference of energy E associated with "orbits-jumping" can be expressed as the difference between the energy of the electron on the initial E_i and the final E_f orbit:

$$\Delta E = E_f - E_i \quad (2.5)$$

Max Planck has deduced that the energy of electrons comprising the electromagnetic radiation is a function of frequency, from where his famous formula comes:

$$\Delta E = h\nu = -R_h \left(\frac{1}{n_f^2} - \frac{1}{n_i^2} \right) \quad (2.6)$$

where h is the *Planck constant* ($6.63 \times 10^{-34} Js$) and ν is the frequency of the

emitted light (Figure 2.1).

2.3.2 Quantum Model of Elementary Particle

This brief look into the physical background should be completed by the fact that Bohr's model of atom assumed that the electron is orbiting the nucleus similarly the Earth is orbiting around the Sun, which violates the Heisenberg uncertainty principle of Quantum Mechanics. This principle states that the position and the momentum of an EP cannot be simultaneously determined with certainty. In particular in quantum mechanics, any elementary particle has the property that the root-mean-square deviation of the position p from the mean $\Delta p = \sqrt{\langle p^2 \rangle - \langle p \rangle^2}$ and root-mean-square of the momentum m from the mean $\Delta m = \sqrt{\langle m^2 \rangle - \langle m \rangle^2}$ multiplied together is never smaller than $\frac{\hbar}{2}$. This is also expressed by the commutator:

$$[m, p] = i\hbar \quad (2.7)$$

The introduction of these unusual properties was required to correctly describe the QM system (sometimes also referenced as a "failure" of the classical Bayesian statistics [You95]) and to allow predicting states of Physical Quantum Systems.

Example 2.3.2.1 Two-Slit experiment

In the two-slit experiment, the dual nature of EP was shown. The experiment consists of the emitter (device firing EP on a screen), of the screen with two

Figure 2.1: Bohr model of the atom (nucleus, orbiting electrons). Shown are light colors respective to the electron orbit transitions.

holes and of the detector. Figure 2.4 illustrates the experimental setting. The system is setup so that the electrons detected by the detector have to travel through the open holes (the screen is thick enough to stop the electrons completely). When only one of the two slits is open and the observer looks at (measures) the projection of fired particles on the detector, the distribution of their locations is proportional to a linear trajectory through the opened slit (particle behavior). The paradox shows up when both the slits are opened.

Figure 2.2 shows the detection screen and the number of electrons measured when either the top or the bottom slit is open. Two curves show the distribution of particles on the detector screen either with top or with the bottom slit open. The thick curve is an expectation of what should be the particle distribution with both slits opened based on the classical probability theory. What appears to be a classical probabilistic distribution of particles with only

Figure 2.2: The measured number of electrons on the detector screen with top or the bottom slit open (thin lines). The expected probability when both slits are open (thick line).

one of both slits open, is transformed to an interference pattern with both slits open (Figure 2.3), not obtainable using classical statistics.

The problem when this measurement was made was to interpret it and to decide whereas EP's travel in space on a straight line (as particles) or if they have the wave properties. The problem was to determine how an EP (electron or photon) has the particle characteristics (mass and speed) when measured and could behave as a wave at the same time? The dual nature problem is solved by the supposition that the EP is a particle while the measurement is performed, and the EP behaves like a wave while not.

According to Figure 2.4 it is not possible to decide whereas a particle traveled through one, two or both slits simultaneously because the measurement does not allows determining it. If a measurement of particles is done on the screen, the result will yield 50% of particles through left slit and 50% through the right slit. The consequence of these observations is the fact that while

Figure 2.3: Results of measurement of particles position when both slits of the screen are open.

Figure 2.4: Schematic representation of the two-slit experiment. Left is the emitter and on the right is the detector (film). In the middle is the barrier with two holes.

recording probabilities of detecting an electron in the interference pattern, probabilities of observation of a given state can be smaller than in standard Bayesian probabilistic model! This implies the following contradictory equation 2.8 [You95].

$$P(x) = P(x|slit_1) + P(x|slit_2) \leq P(x|slit_1) \quad (2.8)$$

where $P(x)$ is the probability of measuring a particle on position x . The solution to this problem was the introduction of the concept of the complex probability amplitudes because such amplitudes can cancel each other. This system is then mathematically a set of functions mapping real physical states from Hilbert space H into a complex space C :

$$\Psi : S \rightarrow C \quad (2.9)$$

where S is the physical space of states and C is complex vector space, respectively. As will be seen later, the functions from the set described by eq. 2.9 are the wave functions that represent non-trivial states of the quantum system. This state of a particle traveling through both slits is defined as the "superposed" state of the system. For one-particle system this superposition is a result of all its possible states that it is measured for. Its complex probability amplitude α is related to the classical probability p of measuring this system in a particular state by $|\alpha|^2 = p$. In a system of n particles (called also

the quantum register), the system constitutes a superposition of $m * n$ states where m is the number of states of each elementary unit of this system.

2.3.3 Schrödinger equation

The general solution for quantum mechanical events is given by the Schrödinger equation:

$$i\hbar \frac{d|\psi\rangle}{dt} = H|\psi\rangle \quad (2.10)$$

with $\hbar$ being the Planck constant and H being the Hamiltonian of the system. A Hamiltonian represents the observable corresponding to the total energy of the system. In particular, the possible observable states are represented by the spectrum of the Hamiltonian [Dir84]. This general equation describes the natural evolution of a quantum system. The $\hbar$ constant can be absorbed into the Hamiltonian H . The Hamiltonian H can for example represent a particle that exists in the infinite one-dimensional potential such as the Simple Harmonic Oscillator (SHO) model with Hamiltonian $H = \frac{p^2}{2m} + \frac{1}{2}m\omega^2 x^2$. The Schrödinger equation for SHO takes the form of:

$$\frac{-\hbar}{2m} \frac{d^2\psi_n(x)}{dx^2} + \frac{1}{2}\beta x^2\psi_n(x) = E\psi_n(x) \quad (2.11)$$

where $V(x) = \frac{1}{2}\beta x^2$ is the potential well with $\omega = \sqrt{\frac{\beta}{m}}$ (and β being the spring constant). In general the solution that one obtains when solving for physical

systems is a solution to eq. 2.10 which is of the form:

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle \quad (2.12)$$

or more clearly as

$$|\psi(t)\rangle = e^{-in\omega t}|\psi(0)\rangle \quad (2.13)$$

with ω being the angular frequency and n being the index of distinct non-degenerate quantum states. From the eq. 2.13 and with respect to previous stipulations (or postulates of quantum mechanics) it can be observed that the resulting state is a distribution of corresponding probabilities p_n over the set of eigenstates n . For more details, this particular problem has all solutions represented by Hermite polynomials, however it is not the focus of this dissertation.

2.3.4 Superposition of quantum states

Let's illustrate the superposition phenomena by work done in [MMKW96]. In this experiment described is the creation of a "Schrödinger cat" state of an atom. The "Gedankenexperiment" of Schrödinger was to place a living cat into a superposition of being alive and dead. These superposed states in QM are described by a wave function. The cat state can be described in these terms as follows:

$$|\Psi\rangle = \frac{|\uparrow\rangle + |\downarrow\rangle}{\sqrt{2}} \quad (2.14)$$

where $|\uparrow\rangle$ and $|\downarrow\rangle$ refer to the states of a living and dead cat, respectively. Once again this situation is not "realistic" in our macro-world, however appropriate for the QM. The interpretation of the general equation 2.14 is that for each measurement of the system described by it there is 50% chance to find the system in state $|\downarrow\rangle$ and a 50% chance to find the system in state $|\uparrow\rangle$. This can be formalized in Dirac's notation as:

$$\Psi = \alpha|\uparrow\rangle + \beta|\downarrow\rangle \quad (2.15)$$

$$= \frac{1}{\sqrt{2}}|\uparrow\rangle + \frac{1}{\sqrt{2}}|\downarrow\rangle \quad (2.16)$$

with

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.17)$$

Where α and β are in general complex amplitudes associated with each measured state, and $|\alpha|^2 = \alpha\alpha^*$ where $\alpha^* = (a + ib)^* = (a - ib)$ is a complex conjugate of α . Thus both states $|\uparrow\rangle$ and $|\downarrow\rangle$ will be observed when measured with probabilities $|\alpha|^2$ and $|\beta|^2$ respectively. This interpretation of the system

defined by (2.16), shows that any quantum system can be represented by a wave function describing all possible states of the system (here we assume two orthonormal states $|\downarrow\rangle$ and $|\uparrow\rangle$) by using complex probabilities (here α and β). The complex probabilities are restricted only by the second equation in (2.17), and the observables of this quantum system are valued in the range of $\{0, 1\}$.

Now, stepping back to [You95], if instead of a cat we consider some of alkali-like ions such as $^{40}\text{Ca}^+$, $^{24}\text{Mg}^+$ or $^{198}\text{Hg}^+$ which do not have a third electronic ground state available for the auxiliary level (thus can be directly used for quantum permutative computing)[MMKW96], the equations (2.16) and (2.17) would represent the wave function of an ion where $|\downarrow\rangle$ and $|\uparrow\rangle$ are two distinct energetic states.

As mentioned earlier in this section (also in Section 2.4), operations on single trapped ions have been already implemented [MMKW96, MML⁺98]. This technique consists in trapping in an Electro-Magnetic field one or more ions and using laser beams setting these ions into certain states. Applying a well-determined sequence of laser pulses on particular ions in the trap, one can achieve such gates as NOT (an inverter of classical logic design). Again the setting of ions is represented by jumps of electron on their orbits and emitting or absorbing quanta of energy. It also allows realizing one, two or even three qubit gates. However, no large-scale quantum computer was yet experimentally created using ion-traps.

2.4 From Quantum Mechanics to Quantum Logic

In the Section 2.3, the example of implementation of a quantum computer was mentioned: the trapped ions in the EM field interacting with laser beams. This has for consequence the changes of states of the ions such as moving from ground state to an excited state by an electron jump from a lower orbit to a higher one. But electrons are more complicated than just quantum energy collectors and their wave functions are more complex because they depend on three parameters beside the principle quantum numbers; the orbital quantum number "l", the magnetic quantum number "m" and the spin "S". While parameters l and m determine the angular dependence, the S determines the internal electron rotation. For our exploration it is only important to know the S number is often used to represent basis states in a Quantum Computation because the hydrogen atom can have only two values $\pm\frac{1}{2}$ of spin. Consequently, using spin rotations for the basis implies working directly with a two-valued quantum logic. Now, considering the spin S value as the basis, a quantum operator on an atom will result in a rotation of the electron spin. Consequently all single qubit operations can be expressed as rotations by certain angles [NC00]. Adding these explanations to the mathematical description implies the fact that our operator on a physical system will yield results in dense complex space C. As moreover we are in an inner-product space H, all states are vectors and operators on them are matrices that represent the evolution of the quantum system.

The adaptation of the physical model to the computing model requires a definition of units. Derived from previous experiments, the units for this logical system will be now called qubits (quantum bits) [Fey85, NC00, Dir84]. The difference between a classical bit and a qubit is mainly the fact that a qubit can represent a superposition of its states. This is different from the states of the classical logic. Assume a system with n qubits. A classical register represents 2^n distinct states while the quantum system is an arbitrary superposition of these 2^n states. Practically it means that while not measured the system can be in one, two, three ... or all of the 2^n states at the same time! Consequently a qubit is defined as a bit of information with quantum properties.

In general, to describe basis states of a Quantum System, the Dirac notation is preferred to the vector based Heisenberg notation. This is mainly because Dirac notation is much more practical than Heisenberg notation for proving facts in Quantum Computing (Heisenberg notation is useful in computer calculations). First let the orthonormal quantum states be represented in the vector (Heisenberg) notation eq. 2.18.

$$\begin{aligned} |\uparrow\rangle = |0\rangle &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ |\downarrow\rangle = |1\rangle &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned} \tag{2.18}$$

Different states in this vector notation are then multiplications of all possible states of the system, and for a two-qubit system we obtain (using the Kronecker product [Gru99, Gra81, NC00]) the states represented in eq. 2.19:

$$\begin{aligned}
 |00\rangle &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} & |10\rangle &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\
 |01\rangle &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} & |11\rangle &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}
 \end{aligned} \tag{2.19}$$

Example 2.4.0.1 Kronecker Product

Similarly for Operators, the Kronecker product exponentially increases the dimension of the space:

$$W \otimes H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{2.20}$$

Figure 2.5: Circuit representing the $W \otimes H$ operation

This operation is shown in Figure 2.5.

Assume that qubit a (with possible states $|0\rangle$ and $|1\rangle$) is represented by $|\Psi_a\rangle = \alpha_a|0\rangle + \beta_a|1\rangle$ and qubit b is represented by $|\Psi_b\rangle = \alpha_b|0\rangle + \beta_b|1\rangle$. Each of them is represented by the superposition of their basis states, but put together the characteristic wave function of their combined states will be:

$$\begin{aligned} |\Psi_a \Psi_b\rangle &= \alpha_a \alpha_b |00\rangle + \alpha_a \beta_b |01\rangle \\ &+ \beta_a \alpha_b |10\rangle + \beta_a \beta_b |11\rangle \end{aligned} \quad (2.21)$$

with α_a and β_b being the complex amplitudes of states of each EP respectively. As shown before, the calculations of the composed state are achieved via the Kronecker multiplication operator. Hence come the quantum memories with extremely large capacities mentioned earlier and the requirement for efficient methods calculating such large matrices.

The interpretation of equation 2.21 is that it represents the state of a qubit system now. It represents the possible states of the "qubit system" more precisely and shows the probability that the qubits will be measured in states

$|\downarrow\rangle$ or $|\uparrow\rangle$. The bases of our model are not unique, other bases such as EPR (Einstein-Podolsky-Rosen) orthonormal bases are also used [EPR35], but for more clarity here we will be dealing only with these simple bases.

Before continuing the explanation of quantum logic synthesis, main properties of quantum computational systems must be mentioned. The wave representation allows the quantum computational mechanism to inherit properties of waves such as superposition, entanglement and measurement. First, however, the principles of multi-qubit system must be introduced.

Example 2.4.0.2 Multi-Qubit System

To illustrate these important properties let's have a look at a more complicated system with two quantum particles a and b represented by $|\psi_a\rangle = \alpha_0|0\rangle + \beta_a|1\rangle$ and $|\psi_b\rangle = \alpha_b|0\rangle + \beta_b|1\rangle$ respectively. For such a system the problem space increases exponentially and is represented using the Kronecker product [Gru99].

$$|\psi_a\rangle \otimes |\psi_b\rangle = \begin{bmatrix} \alpha_0 \\ \beta_0 \end{bmatrix} \otimes \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \alpha_0\alpha_1 \\ \alpha_0\beta_1 \\ \beta_0\alpha_1 \\ \beta_0\beta_1 \end{bmatrix} \quad (2.22)$$

Thus the resulting system is represented by $|\psi_a\psi_b\rangle = \alpha_a\alpha_b|00\rangle + \alpha_a\beta_b|01\rangle + \beta_a\alpha_b|10\rangle + \beta_a\beta_b|11\rangle$ (2.21) where the double coefficients obey the unity (completeness) rule and each of their powers represents the probability to measure the corresponding state. The superposition means that the quantum system is or can be in any or all the states at the same time. This superposition poten-

tially gives the massive parallel computational power to quantum computing.

2.4.1 Simple Projective Measurement

As the states of qubits are vectors with complex coefficients, the real logic state is obtained by measuring the system and observing the result. The measurement process projects the measured qubit onto the set of real valued observables. In other words a quantum system is described by a set of *filters*, each letting through and capturing a particular state. As will be seen later, for logic design this implies that not only one can measure for a set of observables to design a function but also one can use various sets of observables in order to obtain different functions.

Definition 2.4.1 Projective measurement

The measurement of a single binary qubit is described by the overall probability of observing both orthonormal states given by $p(0) + p(1) = 1$. Thus, in a more general way, any ($\frac{1}{2}$ - spin) quantum system is described by:

$$p(m) = \sum_n \langle \Psi | M_n^\dagger M_n | \Psi \rangle \quad (2.23)$$

where $p(m)$ is the probability to measure value m , Ψ is the wave representation of the circuit and M_n is the measurement operator for the value n . From the

above it is simple to derive the complete description of a single qubit circuit as:

$$p(0) + p(1) = \langle \Psi | M_0^\dagger M_0 | \Psi \rangle + \langle \Psi | M_1^\dagger M_1 | \Psi \rangle = 1 \quad (2.24)$$

where

$$M_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [1 \ 0] = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \text{ and } M_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} [0 \ 1] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad (2.25)$$

, where $M_0^\dagger$ and $M_1^\dagger$ are their respective Hermitian conjugates. Because

$$M_0^\dagger M_0 + M_1^\dagger M_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \quad (2.26)$$

, and $M = \sum_k M_k = M_0 + M_1$ (representing all possible outcome values) describes the quantum system completely. $\square$

Example 2.4.1.1 Entanglement and Projective Measurements

Assume the above two-particle vector is transformed using the quantum circuit from Figure 2.6.

This circuit executes first a Hadamard transform on the top qubit and then a Controlled Not operation with the bottom qubit as the target. Depending

Figure 2.6: EPR producing circuit

on the initial state of the quantum register the output will be either $|\psi_a\psi_b\rangle = \alpha_a\alpha_b|00\rangle \pm |\beta_a\beta_b|11\rangle$ or $|\psi_a\psi_b\rangle = \alpha_a\beta_b|01\rangle \pm |\beta_a\alpha_b|10\rangle$. Thus it is not possible to estimate with 100% probability the initial state of the quantum register.

Let $|ab\rangle = |00\rangle$ at level a (Figure 2.6). The first step is to apply the $[H]$ gate on the qubit-a and the resulting state at level b of the circuit is

$$\begin{aligned}
 |ab\rangle &\rightarrow (H \otimes W)|ab\rangle \\
 &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle \\
 &= \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} \quad (2.27)
 \end{aligned}$$

Next the application of the CNOT gate results in:

$$|\psi_a\psi_b\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad (2.28)$$

For an output 0 (on the qubit-a), the projective measurement of the first (topmost) qubit (qubit-a on Figure 2.6) on this stage would collapse the global state (with a single measurement) to the state $|00\rangle$:

$$|ab\rangle \rightarrow \frac{M_0|ab\rangle}{\sqrt{\langle ab|M_0^\dagger M_0|ab\rangle}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T = |00\rangle \quad (2.29)$$

with

$$M_0|ab\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.30)$$

and

$$\begin{aligned}
\sqrt{\langle ab|M_0^\dagger M_0|ab\rangle} &= \sqrt{\frac{1}{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\
&= \frac{1}{\sqrt{2}}
\end{aligned} \tag{2.31}$$

Similarly, the probability of measuring output on the qubit-a in state $|0\rangle$ is:

$$\begin{aligned}
p(0) &= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.32} \\
&= \frac{1}{2}
\end{aligned}$$

If one would look to the output of the measurement on the second qubit (qubit-b), the probability for obtaining $|0\rangle$ or $|1\rangle$ is in this case the following:

$$\begin{aligned}
p(0) &= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.33} \\
&= p(1) = \frac{1}{2}
\end{aligned}$$

Thus the expectation values for measuring both values 0 or 1 on each qubit independently are $\frac{1}{2}$.

If however one looks on the second and non-measured qubit (if the qubit-a is measured, it is the qubit-b, and vice versa) and calculates the output probabilities, the output is contradictory to the expectations given by standard probabilistic distribution such as a coin toss $q = 1 - p$. To see this let's start in the state

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.34}$$

and measure the qubit-a and obtain a result. In this case assume the result of the measurement is given by:

$$|\Psi\rangle \rightarrow \frac{M_0|\Psi\rangle}{\sqrt{\langle\Psi|M_0^\dagger M_0|\Psi\rangle}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.35)$$

Then measuring the second qubit (qubit-b) will not affect the system because the measurement of the qubit-a has collapsed the whole system into a single basis state:

$$|\Psi\rangle \xrightarrow{M} |00\rangle \quad (2.36)$$

The probability for obtaining a $|1\rangle$ on the qubit-b is thus 0 and the measurement on qubit-b (after having measured qubit-a) has no effect on the system at all. This non-locality paradox was first described by Einstein-Podolsky-Rosen work[EPR35] and is known as the EPR paradox. This particular phenomenon is one of the most powerful in quantum mechanics and quantum computing, as it allows together with superposition the speedup of solutions to certain types of problems. Finally, it can be noted that mathematically, the entangled state is such that it cannot be factorized in simpler terms. For example, the state $\frac{(|00\rangle+|01\rangle)}{\sqrt{2}} \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle$ and thus is factorizable. However, the states as those introduced above cannot be transformed in such a manner and are thus

entangled; physically implying that they are related through measurement or observation. $\square$

2.4.2 Density Matrix and POVM

When dealing with measurement and representation of systems that are not completely known it is useful to represent the system using the Density Matrix representation [NC00]. A quantum system $|\psi\rangle$ spanning a Hilbert space on basis states $|0\rangle$ and $|1\rangle$, with p_i the probability of observing value i can be represented as:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \quad (2.37)$$

With $|\psi_i\rangle\langle\psi_i|$ being the outer-product (Kronecker multiplication), ρ is the density matrix and p_i is the probability of observing the given collapsed state $|\psi_i\rangle\langle\psi_i|$. For example, for a system described by $|\phi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$ we

have:

$$\begin{aligned}
 \frac{|00\rangle\langle 00|}{2} + \frac{|01\rangle\langle 01|}{2} &= \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
 \end{aligned} \tag{2.38}$$

As can be expected the eq. 2.38 represents only the vector part of the overall system. To make a complete description one needs to determine precisely the probability of observation of the states $|00\rangle\langle 00|$ and $|01\rangle\langle 01|$. Because the density matrix must describe the system completely, the final form of the eq.

2.38 is shown in eq. 2.39.

$$\rho = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.39)$$

Let M_k be a measurement operator for the quantum state $|k\rangle$ such as $M_{00} = |00\rangle\langle 00|$, then:

$$\begin{aligned} \sum_i p_i \langle \psi_i | M_k^\dagger M_k | \psi_i \rangle \\ &= \sum_i p_i \text{tra}(M_k^\dagger M_k | \psi_i \rangle \langle \psi_i |) \\ &= \text{tra}(M_k^\dagger M_k \rho) \end{aligned} \quad (2.40)$$

The final state of the system in the post-measurement state $|\psi_k\rangle$ is described by applying the given operator to the quantum state. Thus one of the measurement operators from $M = \sum_k M_k$ (as shown in example 2.4.1.1) is used and the final state is represented as:

$$|\psi_k\rangle = \frac{M_k |\psi\rangle}{\sqrt{\langle \psi | M_k^\dagger M_k | \psi \rangle}} \quad (2.41)$$

In density matrix notation this can be expanded to:

$$\rho_k = \sum_i p_i \frac{M_k |\psi_i\rangle \langle \psi_i| M_k^\dagger}{\sqrt{\langle \psi_i | M_k^\dagger M_k | \psi_i \rangle}} = \frac{M_k \rho M_k^\dagger}{\text{tra}(M_K^\dagger M_K \rho)} \quad (2.42)$$

The density matrix is also useful to describe quantum ensemble systems (quantum system constructed from multiple independent subsystems). Such system represents a set of prepared states. For instance, a set of prepared states such that 50% of them is in $|0\rangle\langle 0|$ and 50% in $|1\rangle\langle 1|$ is said to be in a mixed state (a statistical average) (eq. 2.43).

$$\rho_R = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| \quad (2.43)$$

In contrast, a system in a state $|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ is said to be in a pure state (eq. 2.44).

$$\rho_Q = |\phi\rangle\langle \phi| \quad (2.44)$$

The $\text{tra}(\rho^2)$ is also a measure for whether the system is in pure state or in a mixed quantum state. It can be seen, that given ρ_R , the trace $\text{tra}(\rho_R^2) < 1$ (the system is in mixed state) while $\text{tra}(\rho_Q^2) = 1$ because it is a pure quantum state [NC00]. In other words, a pure state is a quantum state that can be represented by a single ket vector (a probability of 1) while a mixed state is a statistical distribution of states. Thus a pure state has a density matrix with a single probability $p = 1$.

Note that we naturally extended the single qubit measurement operators from eq. 2.25 to multi qubit measurement such as M_{00} or M_{01} . This is formally

possible despite the fact that as a result of contemporary measurement technology, only single qubit measurements are allowed and to detect multiple qubit states, synchronous single-qubit measurements must be executed [LBA⁺08].

This distinction can be observed from the logic point of view; quantum operators corresponding to boolean reversible functions will be represented as permutation matrices while quantum operators with probabilistic outcomes will be represented by unitary operators having coefficients (possibly complex) ($\sqrt{|\alpha|^2=0} < |\alpha| < \sqrt{|\alpha|^2=1}$). A similar observation can be made for a density matrix; a density matrix such that $\text{tra}(\rho^2) = 1$ will represent a permutative reversible logic function.

So far in this chapter we have been using the Projective Measurement (illustrated in examples above) however other types of measurement are also possible. Another type of measurement (much more realistic) is the Positive valued-operator measurement (POVM). The importance of POVM is due to the fact that the projective measurement is not well suited to measure for states in non orthonormal bases. Moreover POVM, unlike the projective measurement case, does not always allow to determine the complete state of the system after a measurement (contrary to the case of projective measurement operator $M = \sum_k p_k M_k$) but rather allows to make prediction about the probabilities of the different possible measurement outcomes represented by the density matrix. This is because if one desires to measure two non orthonormal states and with the condition that every quantum system is completely described by $M = \sum_k p_k M_k$, then because the two desired states do not span the complete

underlying Hilbert space, there is a state that when measured nothing can be said about the observed result.

Example 2.4.2.1 Projective Measurement of non-orthonormal states

For example assume we have a quantum system $|\Phi\rangle$ in a Hilbert space spanned by 2 orthonormal basis states $|0\rangle$ and $|1\rangle$. In this setting the projective measurement allows to determine completely the state of the system. Now assume that our basis states are not orthonormal, and we use for measurement the states $|0\rangle$ and $\frac{(|0\rangle+|1\rangle)}{\sqrt{2}}$ [NC00]. The projective measurement operators for these states are shown in equation 2.45.

$$\rho_0 = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \rho_a = \frac{1}{2}(|0\rangle + |1\rangle)(\langle 0| + \langle 1|) = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad (2.45)$$

Looking now to the outcome of the possible measurements by using $|\psi\rangle = \frac{M|\psi\rangle}{\langle\psi|M^\dagger M|\psi\rangle}$, for a initial state $|0\rangle$ we obtain:

$$\frac{\rho_0|0\rangle}{\sqrt{\langle 0|\rho_0^\dagger\rho_0|0\rangle}} = 1 \quad \text{and} \quad \frac{\rho_a|0\rangle}{\sqrt{\langle 0|\rho_a^\dagger\rho_a|0\rangle}} = \frac{1}{2} \quad (2.46)$$

The obtained measurement results in value 1 when measuring using the right measurement basis. However, in the case when the result of measurement is probabilistic (observing state $|0\rangle$ and state $|1\rangle$ with equal probability), the state prior to the measurement cannot be determined with certainty as it could be either of them. $\square$

As can be seen in the above example, the projective measurement defined is under-dimensioned to capture all the features of this system. One solution to this problem is to create projective measurement of a higher dimensional space. The other solution is to use the POVM measurement. That means, that for a quantum system that does not have orthonormal bases, it is still possible to predict the outcome of the measurement.

Definition 2.4.2 POVM

A POVM is a set of Hermitian positive operators such that $\sum_i |u_i\rangle\langle u_i| = I$ as only requirement. $\square$

Example 2.4.2.2 Constructing a POVM

Let $|u_i\rangle$ be a set of non-normalized quantum states such that:

$$\begin{aligned} |u_0\rangle &= \frac{\sqrt{2}}{\sqrt{3}}|1\rangle \\ |u_1\rangle &= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{6}}|1\rangle \end{aligned} \tag{2.47}$$

The non-normalized quantum states (eq 2.47) and similarly to eq. 2.46 the probabilities of outcomes for these states are given by $|\langle u_i|\Phi\rangle|^2$. This means that when the outcome of our measurement generates an eigenvalue of 1, the state of the system is well detected. When the output is probabilistic (after measurement) the initial state of the system cannot be determined with certainty. With such initializations of states $|u_i\rangle$, it is possible to measure for more states that is possible in the orthonormal basis set. Thus let represent the states $|u_i\rangle$ as corresponding density matrices D_i of the observable:

$$\begin{aligned}
D_0 &= \frac{2}{3}|1\rangle\langle 1| = \frac{2}{3} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \\
D_1 &= \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{6}}|1\rangle\right)\left(\frac{1}{\sqrt{2}}\langle 0| - \frac{1}{\sqrt{6}}\langle 1|\right) = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2\sqrt{3}} \\ -\frac{1}{2\sqrt{3}} & \frac{1}{6} \end{pmatrix} \\
D_2 &= I - D_0 - D_1
\end{aligned} \tag{2.48}$$

If the POVM from eq. 2.48 are used to detect the desired states from eq. 2.47, it is easy to see that both $|u_0\rangle$ and $|u_1\rangle$ results in observing D_0 and D_1 with equal probability independently from the initial state (similarly to example 2.4.2.1).

To solve this, one can construct such POVM that each observable D is orthogonal to u :

$$\begin{aligned}
D_{0+} &= \frac{2}{3}|0\rangle\langle 0| = \frac{2}{3} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\
D_{1+} &= \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{6}}|1\rangle\right)\left(\frac{1}{\sqrt{2}}\langle 0| + \frac{1}{\sqrt{6}}\langle 1|\right) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2\sqrt{3}} \\ \frac{1}{2\sqrt{3}} & \frac{1}{6} \end{pmatrix} \\
D_2 &= I - D_0 - D_1
\end{aligned} \tag{2.49}$$

The observables D_{0+} and D_{1+} now allow to distinguish between states u_0 and

u_1 . For instance, when D_{0+} is observed the initial state must have been u_1 ($D_{0+}|u_0\rangle = \{\}$) and when the observable D_{1+} was observed the prior state must have been u_0 ($D_{1+}|u_1\rangle = \{\}$). This can be verified by simple mathematical operations as shown in eq. 2.50

$$\begin{aligned}
\langle u_0|D_{0+}^\dagger D_{0+}|u_0\rangle &= 0 \\
\langle u_1|D_{1+}^\dagger D_{1+}|u_1\rangle &= 0 \\
\text{and} & \\
\langle u_0|D_{1+}^\dagger D_{1+}|u_0\rangle &= 1 \\
\langle u_1|D_{0+}^\dagger D_{0+}|u_1\rangle &= 1
\end{aligned} \tag{2.50}$$

The D_2 represents the uncertainty of the missed measurement and thus each time the D_2 is observed, nothing cannot be said about the system. In other words, for two given non orthonormal states, one can construct a POVM such that for each desired state expressed by D_n $p(D_n) = 1, p(D_m) = 0, \forall m \neq n$.

Thus, well designed POVM allows to distinguish between non-orthonormal states for the price of obtaining no information at all about the system in some cases. It can be noticed that using the POVM operators one will not obtain any information about the state of the system before the measurement. However, POVM allows to determine the probabilities of outcome (quantum state) with certainty.

□

2.5 Quantum Logic, Quantum Gates and Quantum Logic Circuits

2.5.1 Single-qubit Quantum Gates

We are now concerned with matrix representation of operators. The first class of important quantum operators are the one-qubit operators realized in the quantum circuit as the one-qubit (quantum) gates. Some of their matrix representations can be seen in equation 2.51 and they are the "A" linear unitary operators from the Section 2.2.

$$\begin{aligned}
 a) \quad X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & b) \quad Y &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & c) \quad Z &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
 d) \quad H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & e) \quad V &= \frac{(1+i)}{2} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} & f) \quad \text{Phase} &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}
 \end{aligned}
 \tag{2.51}$$

Each matrix of an Operator has its inputs from the top (from left to right) and the outputs on the side (from top to bottom). Thus taking a state $|\psi\rangle$ and an unitary operator H (eq. 2.53)

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.52)$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2.53)$$

the result of computation is represented in equation 2.54.

$$H|\Psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{\alpha+\beta}{\sqrt{2}} \\ \frac{\alpha-\beta}{\sqrt{2}} \end{bmatrix} \quad (2.54)$$

To understand which particular quantum logic operation is executed for each output, the operation from eq 2.54 is broken down to each of the possible input states in eq. 2.55. The first equation from eq. 2.55 shows that when the input state is $|0\rangle$ it is transformed so that when it is then observed the outcome will be 50% of observing the state $|0\rangle$ and 50% of observing the state $|1\rangle$. Similarly for the input state $|1\rangle$, because $p(1) = \frac{1}{2}(\langle 0| + \langle 1|)M_1^\dagger M_1(|0\rangle + |1\rangle) = \frac{1}{2}$.

$$|0\rangle \rightsquigarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

(2.55)

$$|1\rangle \rightsquigarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Observe that while the NOT gate has a unitary matrix that is a permutation matrix, some other gates like the Hadamard and the V gate have unitary matrices that are not permutative matrices. The Hadamard gate is very well known because it is used to create a superposition of states. An example of creating one qubit in a superposition is given in equation (2.55) where for each input either state $|0\rangle$ or $|1\rangle$ the output state $|0\rangle$ or $|1\rangle$ can be measured with a probability of $\frac{1}{2}$.

Observe that the Square-root-of-Not is a unitary transformation creating a complex superposed state (eq. 2.56).

$$\sqrt{X} = V \quad \text{and} \quad \sqrt{X} = V^\dagger$$

where

$$V = \frac{1+i}{2} \begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} \quad V^\dagger = \frac{1+i}{2} \begin{bmatrix} -i & 1 \\ 1 & -i \end{bmatrix} \quad (2.56)$$

Figure 2.7: Schematic representation of Controlled-U gates: a) general structure of single-qubit controlled U gate (control qubit a, target qubit, b) two-qubit controlled single qubit operation, c) single-qubit controlled two-qubit target quantum gate, d) Feynman (CNOT), e) Toffoli (CCNOT), f) Fredkin. a, b, c are input qubits and a', b' and c' are respective outputs.

The V gate has two interesting properties $V \cdot V = V^\dagger \cdot V^\dagger = NOT$ and $V^\dagger \cdot V = V \cdot V^\dagger = I$ and as is shown later, this gate is used to construct the well-known cheapest universal quantum gates.

2.5.2 Multi-qubit and Controlled Quantum Gates

$$\begin{array}{c}
 \begin{array}{c}
 00 \leftarrow \\
 01 \leftarrow \\
 10 \leftarrow \\
 11 \leftarrow
 \end{array}
 U =
 \begin{array}{c}
 \begin{array}{c}
 \downarrow \downarrow \downarrow \downarrow \\
 \left[\begin{array}{cccc}
 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0
 \end{array} \right]
 \end{array}
 \end{array}
 \quad (2.57)$$

The second class of quantum gates includes the Controlled-U gates. Schematic representation of such gates can be seen in Figure 2.7. Gates in Figure 2.7a -

Figure 2.7c represent the general structures for single-qubit-controlled single-qubit, two-qubit-controlled and single-qubit and two-qubit-controlled and two-qubit quantum gates respectively. The reason for calling these gates *Controlled* is the fact that they are based on two operations: first there is one or more control bits and second there is a unitary transformation similar to matrices from equation 2.51 that is controlled. For instance the Feynman gate is a Controlled NOT gate and has two input qubits a and b as can be seen in Figure 2.7 and shown with input and output minterms in 2.57. Thus qubits controlling the gate are called the control qubits and the qubits on which the unitary transform is applied to are called the target qubits.

Figures 2.7d - Figure 2.7f represent special cases where the controlled unitary operator is Not, Not and Swap, respectively. The respective unitary matrices are in equations 2.57, 2.58a and 2.58b.

Equation 2.57 shows that if the input state is for instance $|00\rangle$ (from the top) the output is given by $U|00\rangle = p_{00}|00\rangle = 1 \times |00\rangle$. Similarly for all other possible input /output combinations.

$$\begin{array}{cc}
 (a) & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
 \end{array}
 \begin{array}{cc}
 (b) & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
 \end{array}
 \quad (2.58)$$

The *Controlled-U* gate means that while the controlled qubit a is equal to 0 the qubits on output of both wires are the same as they were before entering the gate ($a' = a$, $b' = b$). Now if qubit a equals to 1, the result is $a' = a$ and $b' = \neg b$ according to matrix in equation (2.51.a). It can be easily verified that the CCNOT (Toffoli) gate is just a Feynman gate with one more control qubit and the Fredkin gate is a controlled swap as shown on Figure 2.7.

A closer look at equations (2.57 and 2.58) gives more explanation about what is described in eq. 2.57: CNOT, eq. 2.58a : Toffoli and eq. 2.58b : Fredkin gates. For instance, equation 2.57 shows that while the system is in states $|00\rangle$ and $|01\rangle$ the output of the circuit is a copy of the input. For the inputs $|10\rangle$ and $|11\rangle$ the second output is inverted and it can be seen that the right-lower corner of the matrix is the NOT gate. Similarly in the other two Controlled

Table 2.1: a). - K-map , b) - LUT

a).

	c	
ab	0	1
00	0	1
01	1	0
11	1	0
01	0	1

b).

abc	f
000	0
001	1
010	1
011	0
100	0
101	1
110	1
111	0

gates the NOT gate matrix can be found.

To summarize, quantum gates can be divided into two major groups: one qubit gates and controlled-U gates. Most of gates are represented by permutation matrices and the gates that cannot be represented by permutation matrices create a superposition of states. Unitary matrices are linear operators modifying complex amplitudes of the input state and thus they affect the probability of measurement of each basis state.

2.5.3 Quantum Circuits and Representation

The representation used to specify quantum functions from the physical point of view is not the most appropriate when designing quantum circuits. In standard approach to Logic Design the function $f(a, b, c)$ is specified as a K-map (Karnaugh Map) or a LUT (Look-Up-Table or Truth Table) (Table. 2.1).

Observe that for the single output function in Table 2.1, the output is balanced; exactly half of the output values are 0 and half of them are 1.

Definition 2.5.1 Reversible Functions

Let $f(.)$ be a completely defined function on $\{0, 1\}^{\otimes n}$ such that for every input vector $i_j \subset I$ (I being the set of all possible input vectors defined over the binary vector of width n) it holds that

$$f(i_j) = o_j \text{ such that } \forall i_j, i_k \subset I; o_j \neq o_k \quad (2.59)$$

Eq. 2.59 represents a one-to-one mapping between input and output vectors defining a reversible function. The importance of reversible functions in quantum computation comes from the fact that every quantum function is reversible up to the measurement. The most obvious example of this phenomenon is the entanglement, where the unitary matrix representing the transformation U is a one-to-one mapping, but once measured this property is lost (example 2.4.1.1). $\square$

Example 2.5.3.1 Reversible Function

The circuit in Figure 2.8a represents the function shown in a K-map Figure 2.8b).

The Table in Figure 2.8b is read as follows: for each logic input combination to the circuit represented as a minterm, the input state is transformed into another quantum state. For instance the quantum state $|100\rangle$ is transformed to state $|000\rangle$ which is denoted as $|100\rangle \rightarrow |000\rangle$.

The representation from the above example is however not appropriate for quantum circuits, as Quantum Unitary operations can yield either determin-

Figure 2.8: Example of representation of a quantum circuit using a quantum K-map. X is the Pauli X rotation or an Inverter.

istic, probabilistic or entangled states. In particular it is important to notice that the entangled states are different from the simple probabilistic states, and thus they require also a special notation.

Definition 2.5.2 Quantum-Reversible Function

Let U be a Unitary transformation in a Complex Hilbert space H^d , such that for every input state $\psi_j \in I$ (I being the set of all possible binary input vectors width n of the quantum register) it holds that

$$U|\psi_j\rangle = |\psi_m\rangle, \text{ such that } \forall |\psi_j\rangle, |\psi_k\rangle \in I; |\psi_m\rangle\langle\psi_n| = 0 \quad (2.60)$$

□

To be more precise, the K-map can be written for output states before the measurement. In such case, the complex coefficients can be written along the output states (Figure 2.9b).

In this case all the quantum complex coefficients are visible. The G_0 and G_1

Figure 2.9: Example of a quantum circuit (a) having a non-permutative unitary matrix (using a quantum K-map (b)). The V is the $\sqrt{X}$ gate.

are two quantum states given by

$$|G_0\rangle = \frac{(0.5 + 0.5i)|0\rangle + (0.5 - 0.5i)|1\rangle}{2} \quad (2.61)$$

and

$$|G_1\rangle = \frac{(0.5 - 0.5i)|0\rangle + (0.5 + 0.5i)|1\rangle}{2} \quad (2.62)$$

respectively. These quantum states are collapsed to observable binary states once measured. It is also possible to set up POV measurement allowing to introduce new output values, otherwise not available in classical computing.

Beside K-maps and LUT's a common representation in classical logic is the BDD (Binary Decision Diagram). A BDD is a canonical representation of a given function that is obtained after the minimization of a Decision Tree allowing for more compact representation of data and functions. In quantum logic synthesis, there are already known decision diagram representations. The most common is Quantum Multi-valued Decision Diagram (QMDD) [MM06], however, because these representations generally minimize the information

available to the observer, we do not use them.

Classical, Reversible and Quantum functions can also be specified only for a subset of input values. In that case they are called Incompletely specified functions. Such specifications of functions are very useful in machine learning, where only a subset of input-output pairs of values is known. This is due to the fact that learning tasks in Machine Learning are dealing with real-world problems that are in general not completely known or understood.

Definition 2.5.3 Incompletely Specified Function

An incompletely specified function is a set K of input-output pairs $(|\psi_j\rangle, |\psi_l\rangle)$ such that $|K| < 2^{|N|}$ for any boolean function. For instance, a 3×1 incompletely specified function is shown in Table 2.2.

Table 2.2: K-map of an incompletely specified 3×1 reversible quantum function before measurement

c	0	1
ab		
00	$ 0\rangle$	$ 1\rangle$
01	—	$ 1\rangle$
11	—	$ 1\rangle$
01	$ 0\rangle$	—

□

Definition 2.5.4 Reversible Function Prototype

An incompletely specified function is a reversible-function prototype (specified by K) if and only if there is such an Unitary transformation in a Complex Hilbert space H^d , $U(.)$ such that for every input state $\psi_j \in I$ (I being the set

of all possible binary input vectors width n of the quantum register)

$$U|\psi_j\rangle = |\psi_m\rangle, \forall |\psi_j\rangle \in K \subset I; |\psi_m\rangle = |\psi_l\rangle \quad (2.63)$$

This means that a incompletely specified function is a set of such input-output states (logic values) that must be realizable by a reversible logic function (a permutative unitary matrix). Thus, such incompletely specified function is realizable as a reversible function. $\square$

To represent incompletely specified functions the K-maps or LUTs can be used. However, as it will be seen later, there are various ways how to understand and represent the unknown values that are also called don't cares. A common way of representing such unknown values is shown in Table 2.2, where the symbol '-' represents the don't care value.

Another method of representation is the shortened form. For instance the function defined in Table 2.2 can also be represented as a vector of output values in the natural order of the input values combinations (minterms): $f = [f(0), f(1), \dots, f(7)] = [0, 1, -, 1, 0, -, -, 1]$. To guarantee reversibility one can also use the output values of the whole circuit, thus a three-bit circuit will have outputs in the range $[0, 7]$. The above vector representation can be transformed to $f = [0, 1, ---, 5, 6, ---, ---, 3]$ and a possible result will have to contain exactly one of the possible output values. For instance $f = [0, 1, ---, 5, 6, ---, ---, 3] \Rightarrow [0, 1, 4, 5, 6, 7, 2, 3]$.

Finally, the don't cares can specify a whole minterm or only a part of it. For

instance, a 3×3 incompletely specified reversible function shown in Table 2.3 shows an example of don't cares present in some output states but only on some selected bits.

Table 2.3: K-map of an incompletely specified 3×3 reversible quantum function before measurement with don't cares within single minterms

c	0	1
ab		
00	$ 010\rangle$	$ 001\rangle$
01	$ - 10\rangle$	$ 1 - 0\rangle$
11	$ - 0 - \rangle$	$ 1 - - \rangle$
01	$ - - 0\rangle$	$ - - 1\rangle$

For quantum-reversible function it will be shown later in this dissertation that it is possible to design circuits that not only satisfy the above criteria, but also have probabilistic outputs (on certain or all qubits) along with the deterministic ones. This is done by either specifying the output probabilities of output states or by designing particular measurement operators detecting the specific quantum states.

For instance. Let the output be able to take symbolic quantum state values such as $\{0, 1, V_0, V_1\}$. In order to distinguish by measurement between states V_0 (eq. 2.64) and V_1 (eq. 2.65), one would use a set of basis states to create projective measurement operators. These basis states can be mixed with measurement operators for states $|0\rangle$ and $|1\rangle$ to create POV measurement:

$$|V_0\rangle = \frac{1+i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1+i}{2} \begin{bmatrix} 1 \\ -i \end{bmatrix} \quad (2.64)$$

and

$$D_0 = |V_0\rangle\langle V_0| = \frac{1+i}{2} \begin{bmatrix} 1 \\ -i \end{bmatrix} * \frac{1-i}{2} \begin{bmatrix} -i & 1 \end{bmatrix} = \frac{1}{2} \begin{pmatrix} i & 1 \\ 1 & -i \end{pmatrix} \quad (2.65)$$

$$|V_1\rangle = \frac{1+i}{2} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} * \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1+i}{2} \begin{bmatrix} -i \\ 1 \end{bmatrix} \quad (2.66)$$

and

$$D_1 = |V_1\rangle\langle V_1| = \frac{1+i}{2} \begin{bmatrix} -i \\ 1 \end{bmatrix} * \frac{1-i}{2} \begin{bmatrix} 1 & -i \end{bmatrix} = \frac{1}{2} \begin{pmatrix} -i & 1 \\ 1 & i \end{pmatrix} \quad (2.67)$$

. It can be easily verified that a qubit in state $|V_0\rangle$ is detected with the result of measurement D_1 and the quantum state $|V_1\rangle$ is detected as the result of measurement D_0 (see definition 2.4.2). These operations are also represented in Figure 2.10.

Figure 2.10: Schematic representation of detecting the quantum states V_0 and V_1 using the density matrices D_0 and D_1 .

Thus, a quantum reversible function can be generated such that the outputs include both binary basis states as well as quantum states such as those detected by D_0 and D_1 . Observe, that this approach is different than measuring quantum states and obtaining a probability distributions of orthonormal states

$|0\rangle$ and $|1\rangle$. Moreover, it is a natural extension of the presented ideas that other states such as those based on measurement in circuits that use gates V , $\sqrt{V}$, $\sqrt[4]{V}$, etc can be used to allocate the don't cares in incompletely specified reversible functions.

2.5.4 Quantum Circuits and Sequential Logic

2.5.4.1 Classical vs. Quantum Circuits representation

In classical logic design, one deals with physical elements (CMOS, Transistor Level) that need to have been given a certain input state, that propagates through the network of interconnected gates (logic elements) and generates output after the propagation delay τ . The output will be held as long as the input is held as well. This means that one can describe a reductionist classical circuit on a 2-dimensional plane. The Y axis describes the space (representing the width of the circuit) and allowing for parallel processing on the level of gates. The X axis represents time and space, as it represents the length of the circuit (propagation delay). This can be seen in Figure 2.11.

In quantum circuits this representation must be however modified in order to be correct. Assume a quantum register of width w . While the width of a quantum circuit still corresponds to the Y axis from the classical circuits, the X dimension does not mean the space component anymore. Unlike in the classical case, the logic operation might not be represented by a set of interconnections and logic elements, but instead it is represented by a set of

Figure 2.11: Space-time representation of a classical logic circuit. Time flows from left to right along the X axis (Y axis represents space dimension), unless a vertical interconnection is made in which case time also flows along the Y direction, following the direction of arrows.

localized EM pulses sequentially emitted on the same set of qubits (Figure 2.12).

Observe that this particular representation allows to describe parallelism and serial operation in quantum computing. In each time slice (called t_0 , t_1 , etc.) in Figure 2.12 a set of fully parallel operations on three qubits is represented (such as CNOT and U_1 executed in parallel in slice t_3). In contrast, each time slice is a serial operation with respect to the whole sequence representing the quantum circuit (this is illustrated by the sequence t_0 , t_1 , t_2 , t_3 in Figure 2.12).

The validity of this particular representation is based on the NMR or the Ion Trap spin *state model*. For instance in solid-state quantum computing, the individual qubits can transmit information between neighbors allowing quantum synthesis to be effectively represented by a two dimensional space-time grid. Moreover, as will be discussed in details later, the computing procedure

on a quantum circuit is a sequence of the following operations:

1. initialize the whole quantum register to a desired initial state
2. apply the transformation U
3. measure the desired qubits and observe the result.

Despite the different protocols of quantum computing, the main point for this dissertation is the fact that while in classical logic there is an actual flow of particles through the gates, in quantum computing the gate is dynamically created by EM pulses that create a new quantum state. The state remains unchanged until either another logic operation is applied, the circuit is initialized, the circuit is measured or external events (noise, decoherence) perturb the quantum state.

Figure 2.12: Space-time representation of a quantum logic circuit without measurement. Time flows from left to right along the X axis while space is represented by the Y axis. Remark that the physical size of this circuit is equal only to its width.

2.5.4.2 Quantum Circuit, a natural shift-register.

The conservation of the quantum state implies also the fact that a quantum circuit can be seen as a state machine in a particular state; i.e. after being initialized to $|\phi\rangle$ the state is now $U|\phi\rangle$, with U being the circuit unitary transformation obtained by the sequence of EM control pulses. This equivalence (between quantum circuit and state machine) overlooks one of the main problems in FSM design; the state assignment problem. For instance, assume a FSM with states $Q = \{q_0, q_1, q_2, q_3, q_4\}$ and a state transition function $\delta(q, s) \rightarrow q'$, the problem is to find such state assignment that would minimize the functional logic (the state transition function). In this case, there are $5 \leq 2^3$ states the number of distinct state assignments is $\binom{8}{5} = 6720$. Thus synthesizing a single state machine with a given assignment does not directly addresses the problem of finding the best state assignment, but allows to synthesize machines with the same approach for both quantum circuits and finite state machines.

Thus, a given quantum circuit can be considered as a quantum finite state machine (with determined state encoding), where each iteration of the unitary transformation U on $|\psi\rangle$, will generate the sequence of states given by $|\psi'\rangle = U|\phi\rangle$ starting in the initial state $|\psi_0\rangle$. For instance, the quantum circuit from Figure 2.12, the top qubit (qubit a) can be considered as the FSM internal state qubit, the qubit b as input, and qubit c as output. In such a case, the computing procedure is described by the following steps:

1. initialize the whole quantum register to a desired initial state only once
2. apply the transformation U representing the desired function f
3. measure the output qubit(s) and observe the result.
4. initialize one input qubit (all next initializations of the qubit register only affect the input qubit.)
5. go to 2

Example 2.5.4.1 Simple Quantum Gate as a Finite State Machine

Consider the Feynman gate described by the function shown in Table 2.4. Traditionally the function described by this gate is a change of the values

Table 2.4: K-map of the Feynman gate

b	0	1
a		
0	$ 00\rangle$	$ 01\rangle$
1	$ 11\rangle$	$ 10\rangle$

between two qubits, however the implementation as a FSM allows a different point of view. In this case, assume that the FSM is constructed such that one of the qubits is the input (as described above, it is initialized at each computational step) and the second qubit represents the state. Moreover, a classical controller is required to control the overall functioning of the FSM implementation.

Figure 2.13b shows the CNOT gate built as a FSM; the qubit a represents the input qubit, the qubit b represents the state qubit. The whole setup

is controlled by a classical computer, that can either initialize the quantum register, perform the quantum computation or measure the desired qubits. This setup allows to initialize the input qubit a while preserving the state qubit b. This is also shown in Figure 2.13a where the symbols I_0 or I_1 represents the initialization of the input qubit a and C represents the computation phase.

Figure 2.13: (a) - The state diagram for a FSM using a single Feynman Gate, (b) - schematic representation of a FSM built according to the protocol described above.

□

For more details, some models of quantum state machines are discussed in Appendix B.

As the final point of this section let us discuss the implications that the previous descriptions bear for the logic synthesis:

- Quantum circuits are represented as unitary matrices. The computation includes the initialization of the whole quantum register before computation on any input state. This corresponds to setting the binary minterm as a state of the quantum register. The application of U generates the

output for the given input state. Thus a quantum circuit is directly represented by the unitary transform U .

- Finite State Machines require a register to store the internal state; quantum computing allows naturally to store the complete state (the input signals, the internal state signals, the output signals) during the computation process. Thus using a Unitary transform U allows to represent one of many possible realizations of a given quantum FSM. This FSM has its behavior described by a set of states and a state transition and output function given by the unitary matrix U (both states and the state transition function must be quantum realizable).
- Thus synthesizing Unitary transformations for circuit design, is a quantum equivalent of classical synthesis methods of both combinational logic and state machines.
- The synthesis of a quantum circuit can be executed with respect to the observable output (after the measurement), with respect to the unitary matrix representing the quantum circuit (before the measurement) and with respect to new observable values on the circuit outputs (discovery of circuits generating novel output states) after measurement.

Because of this conceptual difference between various models of quantum computing, the quantum circuit model used in this dissertation is seen as a Unitary matrix, that is built by a sequence of control pulses and operating on a single quantum register. We present logic synthesis methods that: (a)

include the measurement as one of the computational elements, (b) synthesize a circuit without taking into account the measurement and (c) synthesize simple quantum FSM's.

3 QUANTUM LOGIC SYNTHESIS AND ALGORITHMIC SEARCH

3.1 Introduction

Logic minimization is a well known area of computer engineering and in this dissertation various new research aspects related to search, automated synthesis and minimization of quantum circuits are discussed. In Quantum Logic Synthesis, the methods used are directly related to the representation that is being applied and different approaches are used while synthesizing FSM's, Logic Circuits, Behaviors or Quantum Cellular Automata. For instance, to synthesize a FSM using evolutionary approach, the most prominent method includes Genetic Programming [Koz92, Koz94] while the synthesis of boolean logic functions or circuits has mainly been done using Genetic Algorithm or more algorithmic methods such as composition or spectral synthesis [SBM05a, SBM05b, Mil02, MMD06, PARK⁺01, KPK02].

In this chapter introduced are concepts of quantum logic synthesis with respect to quantum primitives and their costs. We describe a general methodology for synthesis of quantum circuits. Various heuristics are studied on the functional level in order to demonstrate logic synthesis methods used for Machine Learning (Chapter 7). These techniques are used later on in the Cynthea robot learning procedures (Appendix C). The described concepts introduce the cost of quantum gates used in our synthesis methods and in particular we analyze

the quantum inductive bias on the logic synthesis of circuits that can be used in the control of behavioral robots using inductive machine learning.

3.2 Previous research on automated synthesis of quantum circuits

The search for smaller, cheaper and ideally optimum circuits in quantum and reversible logic led to a set of gates and circuits commonly used as universal minimal primitives for logic synthesis [MD04, BBC⁺95, Per00, SD96, HSY⁺04]. There are several properties that are being searched for and some of them are: universality, low realization cost, technology specificity and good synthesis properties. In general, the goal is a sum of the mentioned sub-goals with a various degree of importance for every single one of these goals. However, depending on the complexity of the defined problem, it is also required to precisely specify the partial goals and explore them individually.

It was shown by [DiV95, Per00, DiV95, MML⁺98, SD96] that all gates with more than 1 qubit could be build using only one-qubit and selected two-qubit primitives. A big challenge is to build the basic possible gates such as Fredkin [SD96, LPG⁺04] or Toffoli having the smallest cost for a given technology. According to the description of quantum logic in Chapter 2, it is clear that logic synthesis of quantum circuits consists in finding compositions of primitive gates such that their resultant matrix is equal to the specification unitary matrix. This problem can be seen in an analogy to designing classical logic circuits from basic logic gates using a specification in form of a Karnaugh

Map. As was shown in [LPG⁺04] the synthesis of quantum circuits is a non-monotonic process and consequently it is hard to use automated techniques to quantum circuit synthesis without relying on some heuristics. Also as can be implied from matrices representing gates or circuits, their dimensionality grows exponentially with the number of qubits. For example a circuit with 3 qubits will be represented by a matrix of 2^3 by 2^3 (64 elements) while a circuit with 5 qubits will have a matrix of size 2^5 by 2^5 (1024 elements). Each element of such a matrix is in general a complex number and consequently the calculation of the matrix may in the worst case demand also an exponential time. Moreover, in quantum logic synthesis all circuits can be composed in infinitely many ways using quantum gates and without adding more qubits. In other words, a circuit given by a Unitary Transformation U , can be realized either from a minimum number of gates or can be realized in infinitely many circuits of various costs; the more component gates available as the input set, the more solutions to the synthesis are possible. Thus the problem of minimization in Quantum Logic Synthesis is not only a problem of exponentially expanding solution space with the size of the circuit but also finding the minimal set of gates allowing a potentially minimal solution.

Most of known synthesis techniques are either for a small number of qubits only, for a small number of gates or for certain specific constrained logic families of functions (such as reversible or linear functions). Thus despite some already reported results [MD04, LPG⁺04, LP02, YSHP05, YSPH05, SBM05a] there is no general method to synthesize larger than 2-qubit quantum circuits using

Figure 3.1: Schema representing the tree of relations between synthesis approaches for completely specified and incompletely specified quantum/reversible functions as used in this dissertation.

quantum non-permutative primitives. Some of the methods are adapted from Reversible logic synthesis and have been used mainly for synthesis using the CNT set of gates (NOT, Feynman and Toffoli) or similar libraries not allowing to use the entire power offered by the quantum circuits and quantum logic. There exists also a small set of various new libraries of gates for quantum logic synthesis [MD04, BBC⁺95, SD96, LP02, YSPH05]. However there is no proof that any of them has the minimal quantum realization cost with respect to all circuits that can be build from the given functional specification.

3.3 Function Classification for Logic Synthesis

Before discussing various techniques in QLS, let us characterize the problem space we are studying (Figure 3.1). On the top are the reversible/quantum synthesizable logic functions. It is assumed that these functions are either reversible by default or are made reversible (by adding ancilla bits). The com-

pletely specified logic functions represent a class of synthesis problems that is well known in industry and various approaches have already been applied and explored. In this dissertation the focus is mainly on the incompletely specified functions, as defined by Definition 2.5.3. The interest in these functions is mainly based on the facts that a) - incomplete specifications allow to search for novel circuit realizations of functions or automata, b) they can be synthesized for various learning biases (Chapter 7) and c) they are the closest representation of logic functions that can be used in inductive machine learning (Chapter 7).

3.4 Representation

There are two methods of designing and drawing quantum/reversible permutative circuits.

In the first method one draws a circuit from gates and connects these gates by standard wires. This method is similar to classical circuit design, but the used gates are reversible or quantum. The rules to design a reversible circuit using this approach are the following: (1) no loops allowed in the circuit and no loops internal to gates, (2) fan-out of every gate is one. These rules preserve the reversible characteristic of gates thus the resulting circuit is also completely reversible. Next, when a circuit is drawn, the gates are placed on a 2-dimensional space and the connections between them are routed. Every crossing of two wires in the schematics is replaced with the quantum Swap

gate making the schematics planar, which means, no more two wires intersect in it. Also, it is often needed to add ancilla bits initialized to constants. This method is not practical with respect to small width of quantum registers in modern quantum technologies. The schematics is thus rewritten to a quantum array notation used in this dissertation. It is relatively easy to transform a quantum array to its corresponding unitary matrix, as will be illustrated in the sequel. The approaches that use this first design method of reversible circuits are closer to those of the classical digital CAD where the design stages of logic synthesis and physical (geometrical) design are separated. Unfortunately not much is published on this approach and using this methodology can lead to circuit with very wide quantum registers.

The second design method for quantum circuits is to synthesize directly the quantum array of a circuit that was initially specified by a unitary matrix (or a set of functions for desired outputs). This method is executed without involving additional graph-based or equation-based representations. The synthesis is conducted by one of two approaches:

- composing matrices of elementary gates in series or in parallel until the matrix of the entire circuit becomes the same as the specification matrix [LPG⁺03, LPMP02, MD04, Rub00],
- decomposing the specification matrix of the entire circuit to parallel and serial connections of unitary matrices until all matrices correspond to matrices of elementary gates directly realizable in the given technology

[SBM05a, MD04, MDM05, KP06, HSY⁺04, HSY⁺06]. A more mathematical method was used by Yang et al.[YSH05, YSPW05] to analyze group properties of quantum unitary operators and deduce the set of universal quantum gates.

In another synthesis variant, called the approximate synthesis, it is not required that the circuit specification matrix and the matrix of composed gates are exactly the same. They can differ by small allowed values or/and differ in some matrix coordinates only [SBM05a].

In Quantum Logic Synthesis some of the difficulties are the lack of general model for synthesis, heuristics are not well known and when I started to work on this dissertation there were no counterparts in quantum logic of such familiar notions of classical logic CAD as Karnaugh Maps ¹, prime implicants or reductions to covering/coloring combinatorial approaches. Therefore to explore Quantum Gates and QLS some authors turned to evolutionary algorithms as the fast prototyping methods for quantum arrays [WG98, Rub00, LP02, LPG⁺03]. These approaches seem to be good for introductory investigations of the solution space and its properties, with the hope that by analyzing solutions we will learn more about the search space and ultimately create more efficient algorithms based on the acquired knowledge.

¹ K-Maps were first used in quantum circuit synthesis in [LP07]

3.5 Quantum-Based Synthesis: useful quantum circuits synthesis problems

It is well-known that the EPR circuit [NC00, Gru99] composed of a Hadamard gate and a Feynman gate realizes entanglement. In an extended circuit the Hadamard gate can be controlled, which means that when controlled with signal $|0\rangle$, the EPR circuit changes to a single Feynman gate and the entanglement is removed, thus the circuit's behavior becomes deterministic. Similarly the controlled Hadamard and Controlled Square-Root-of-Not (CV) gates can be used as sources of superposition and randomness. Such circuits find applications as possible robot controllers [RFW⁺07, LP07].

Figure 3.2: Toffoli gate realized using 2×2 controlled quantum gates. When used as a quantum robot controller, signals a , b and e can come from touch, sound or other sensors and outputs P , Q and R through measurement units go to motors or other actuators.

The realization of the reversible Toffoli gate (Fig. 3.2) using Controlled-NOT ($CNOT$), Controlled- V (CV) and Controlled- $V^\dagger$ ($CV^\dagger$) gates [BBC⁺95, HSY⁺06, NC00] is another source of inspiration to create quantum circuits because it shows that a deterministic behavior of a permutative (classical reversible) circuit is created using truly quantum gates (such as CV) that operate in Hilbert Space and with intermediate signals that are superposed [NC00]. By truly quantum gates we understand those that their unitary matrices are

not permutative.

If we would thus measure the data path signal in the lowest qubit in Fig. 3.2 in the middle of this circuit, after two *CV* gates controlled by inputs *a* and *b* respectively, the behavior would be deterministic for some input signal combinations and probabilistic for other combinations, leading to very interesting behaviors of robots such as Quantum Braitenberg Vehicles [RFW⁺07] controlled by this circuit.

Even more complicated binary quantum circuits (with permutative unitary matrices) can be composed from gates that are the controlled Pauli X rotations by angles π/k where *k* is a power of two. This leads to gates such as NOT - 180° rotation, square-root-of-not - 90° rotation, fourth-order-root-of-not - 45° rotation, etc. Gates that rotate by $k * (2\pi/3)$ where *k* is an integer are used in ternary quantum logic. These all rotation gates can be controlled by arbitrary quantum states [MMD06]. When the resultant signal in the data path bit (the controlled qubit) is an eigenvalue of the unitary transformation(s), the behavior is deterministic. When it is not, the behavior is probabilistic according to the rules of quantum measurement [NC00, Gru99]. This means that a system in a superposed state, when measured, collapses to one of the possible observables given by the measurement operator. This way, a circuit can be designed from a set of examples corresponding to the care minterms of a truth table. For instance, referring again to Quantum Braitenberg Vehicles, value 0 may correspond to sensor conditions when we want our robot to turn left, and value 1 to the true minterm of input variables (a positive example) when the robot

should turn right. Based on his design goals the designer specifies examples of robot behaviors as input-output pairs. The software induces behaviors for all other input states that are possible.

3.5.1 Cost of quantum circuits

Various cost models have been used in QLS to date. Table 3.1 presents a summary of various costs arranged from the highest level to the lowest one (Pulses, Rotations). The variation of the costs is respective to the synthesis level and can be viewed as a transition cost from Reversible Logic Synthesis to QLS. At the top of Table 3.1 there is a High Level cost of quantum primitives that has a direct correspondence to reversible gates. In the bottom row there is the cost expressed in unitary pulses, thus closest to the real quantum cost. Such gates are uniquely quantum and thus they offer simplifications that cannot be achieved on higher levels. In more details, the first column of the Table 3.1 represents the cost category/level, the second columns shows some of gates used on this level of logic synthesis. The third column specifies the cost specifications, for instance the first row has a cost increasing with the number of controls per logic gate as well as with the number of used gates. Thus, the more controls a gate has the more expensive and difficult it is to realize. The fourth column shows how the overall cost grows; again in the first row it is shown that one can take single-qubit controlled-Not and two-qubit controlled-Not having the basic cost of one, while the cost grows with additional control bits.

Table 3.1: Cost models

Cost Name	Example Gates	Cost Specification	Cost Values
High Level	CNOT,CCNOT,etc	cost per control cost per gate	1,1,2,etc
Standard	W CNOT,CCNOT	cost per wire cost per control	1 2,3,etc.
Restricted	W CNOT,SWAP	cost per wire	1 2,6,etc.
Low Level	$R_x(\phi), I_{ZZ}$	cost per pulse	1,2

Such a reduced cost can be observed for the SWAP gate. In general it is assumed that the SWAP gate is made from three Feynman gates, and thus according to Figure 3.11 it would have a cost of $5 \times 3 = 15$ EM pulses. However because of the nature of the pulses it is possible to combine consecutive pulses on the same axis of the Bloch sphere together and thus minimize the pulse cost to 11 [LLK⁺06]. In this model the cost is calculated according to the bottom line; for instance the $R_x(\pi)$ single qubit rotation has the same cost as rotation $R_y(-\frac{3\pi}{4})$. Moreover, adjacent quantum gates located on the same qubits can as well be combined. This will be shown in Chapter 5. For synthesis this implies that for a particular technology various rules will constrain the search and thus will allow to adjust the cost functions².

Definition 3.5.1 Cost of a Quantum Circuit

$cost = f(Qc_U) = \sum_j U$ is the sum of the costs of each operation used for computation in the circuit. □

² Moving pulses and reducing the sequences of pulses is also considered a useful heuristics for quantum circuits synthesis.

Definition 3.5.1 shows a simple positive cost function monotonic in j .

Figure 3.3: Structure of the Peres gate built from 2×2 quantum primitives

Example 3.5.1.1 Quantum Logic Primitives

There already exists a popularly used library of quantum primitives that is applied by the current quantum logic synthesis algorithms. These primitives are directly derived from atomic operations in quantum mechanics and constitute the basis for logic operations in Quantum computing. In general, these gates are from the following group: Wire, Inverter (Pauli X), Pauli Y, Pauli Z, Hadamard, Feynman, CV , $CV^\dagger$, Peres, Fredkin and Toffoli. In different technologies such as NMR or Ion Trap these gates have different costs and their optimal realizations are not yet established due to the fact that quantum computing is still only at its beginning. From the synthesis point of view, different approaches based on various parameters can be taken to calculate the "total quantum gate cost" of the quantum circuit which estimates the real realization cost of a quantum circuit, that is very much dependent on particular technology or even equipment. We are thus interested only in the total gate cost which we will call the Quantum Cost for short.

The simplest method is to calculate the total number of 2-qubit primitives

Figure 3.4: Structure and minimization stages to reduce the cost of the Fredkin gate. The top diagram shows the circuit non minimized. The middle diagram represents the circuit with permuted gates to allow forming larger blocks. The bottom diagram shows the minimized circuit for which the total quantum cost is calculated [LPG⁺04].

[SD96, LPG⁺04]. Using this method the Peres gate has the cost of 4 (see Figure 3.3), the Toffoli gate (see Figure 3.8) has the cost of 5, and the Fredkin gate has also the cost of 5 (see Figure 3.4). $\square$

3.5.2 The Size of Quantum circuits

Beside the cost, a quantum circuit can be described by a size. However, in order to allow precise definition of the size, first some required concepts need to be defined.

Definition 3.5.2 Quantum Circuit Primitives

A quantum circuit primitive is the smallest unit of the synthesis process. As such, a quantum primitive does not represent a unique gate but rather the smallest unit used to build quantum circuits in a given synthesis model. $\square$

Definition 3.5.3 Quantum Circuit Block

A block of Quantum Circuit is another Quantum Circuit or a Quantum Gate of arbitrary width but smaller than n (in which case the block becomes a segment). Also, a set of quantum gates is a block only if the contained gates are closed; i.e. the gates are fully enclosed inside of the block (no input or output vertically). Example of blocks of gates are shown in Figure 3.5. $\square$

Figure 3.5: Example of Blocks of a Quantum Circuit.

With respect to the definition of block, a Segment of a Quantum circuit is defined by:

Definition 3.5.4 Quantum Circuit Segment

A Segment of Quantum Circuit is another Quantum Circuit or a Quantum Gate of width n , such that it is built only by using Kronecker product between its component gates. $\square$

For instance, the three-qubit quantum circuit from Figure 3.6 can be separated into three segments, each of width three and each being built only by using the Kronecker product. Observe, that such definition then allows to build the quantum circuit by using standard matrix product and a set of segments.

Figure 3.6: Example of Segmentation of a Quantum Circuit. In this case, the circuit is built as a serial composition of three parallel segments a, b and c.

Figure 3.7: Example of Segmentation of a Quantum Circuit. In this case, the circuit is built from three parallel segments a, b, c and d.

The size of the circuit can now be simply defined as:

Definition 3.5.5 Circuit size

The size of a Quantum Circuit is equal to the minimal number of segments that a circuit is described by. $\square$

For instance the circuit from Figure 3.6 has size 3 while the circuit from Figure 3.7 has size 4 despite of having the same cost.

Finally, with the above definitions the overall process of Quantum Logic Synthesis is described in Definition 3.5.6.

3.5.3 Quantum Logic Synthesis

Definition 3.5.6 Quantum Logic synthesis

The Quantum Logic Synthesis problem is to find the circuit that has the minimum value of the Quantum cost (whatever the definition of this cost) for a given truth table, K-map or other specification.

Let $|\psi\rangle$ be the quantum register and let G be a set of single-qubit and two-qubit unitary operators on complex Hilbert space H^n . The process of synthesis of an appropriate quantum circuit finds the unitary matrix U such that the following relation:

$$U|\psi\rangle \rightarrow |\psi'\rangle \quad (3.1)$$

is satisfied for every defined care pair (ψ, ψ') of input-output. Therefore for each such pair the probability of obtaining the output state from the pair is 100%.

In another synthesis variant, the synthesis process can be expressed as a minimization of the given reversible or quantum-reversible function with respect to

- the size of the circuit s_{Qc} (the number of elementary operators used),
- the width of the circuit w_{Qc} ,
- and the error e with respect to the function to be designed as a circuit.

Thus, the QLS process can be written as:

$$S_{HN}(n, G) \xrightarrow{\min(V(s,w,e))} Qc \quad (3.2)$$

where $V(s, w, e)$ is the function to be minimized during the synthesis process. This functions represents the overall cost of the circuit constructed using gates from set G (a size s of the circuit), with a width w of the circuit and with a given error e . $\square$

No synthesis methods are known that would formally consider the trade-off between the width of the quantum register and the quantum cost.

Constructing universal gates from smaller primitives is only one of several goals of quantum logic synthesis. As gates are represented by matrices, an infinity of combinations exists to represent any of the quantum gates. Also it follows logically that a gate can be easily invented by just creating a matrix for a particular function. However, as each gate is in fact one atom or a group of them, a quantum gate (matrix) must be in principle realized in accord with physical laws of EP's. But their costs may differ dramatically.

Reversible functions such as Toffoli or Fredkin are defined as gates and used as synthesis concepts for convenience, but all quantum gates with more than 2 qubits are practically not gates but circuits. This restriction is due to the fact that the state of the art quantum computers allow at present to build only one-qubit and two-qubit gates (the rotation gates and the interaction gates

J_{zz} - Figure 3.11).

The search techniques in quantum synthesis can be mainly split into two streams: algorithmic and heuristic. This is because there are still no tools available that would allow systematic and theory-grounded research in this area. Results from manual or human-based heuristics can be found in [MD04, BBC⁺95, Per00, SD96] while recent publications [LPG⁺04, LP02, Rub00, WG98] show algorithmic rediscovery of the already known gates. Also some interesting circuits have been discovered and proved optimal by automated processes in [MD04, HSY⁺04, YSHP05].

More relevant than in classical circuits synthesis, in quantum circuit synthesis, the technology influences the synthesis to a much higher extent by specifying the primitives to be used in the synthesis. This can be seen in the fact that various quantum technologies use different sets of single qubit rotations as well as different types of interaction gates (in this dissertation we restrict our interest to the popular in NMR J_{zz} interaction gates). Thus every specific variant of quantum technology will specify to a much higher extent what type of gates and single qubit primitives are available for logic synthesis. This is due to the fact that most of the quantum technologies are still only at the experimentation level and thus no standards for primitives, cost or technology-specific constraints have been established.

Figure 3.8: Structure of the Toffoli gate

3.5.4 NMR-based Quantum Logic Gates

The NMR (Nuclear Magnetic Resonance) technology approach to quantum computing [Moo65, PW02, DKK03] is the most advanced quantum realization technology used so far, mainly because it was used to implement the Shor algorithm [Sho94] with 7 qubits [NC00]. Yet other technologies such as Ion trap [DiV95], Josephson Junction [DiV95] or cavity QED [BZ00] are being used. The NMR quantum computing has been reviewed in details in [PW02, DKK03] and for this dissertation it is important that it was so far the NMR computer that allowed the most advanced algorithm (7 qubit logic operation) to be practically realized and analyzed in details. Thus it is based on this technology that the constraints of the synthesis are going to be established for the cost and function evaluation. Some prior work on synthesis has been also already published [LLK⁺06] and few simple cost functions have been established.

For the NMR-constrained logic synthesis the conditions are:

- Single qubit operations: rotations R_x, R_y, R_z for various degrees of rotation θ . With each unitary rotation (R_x, R_y, R_z) represented in equation 3.3.

$$\boxed{\text{X}} = \boxed{iR_x\left(\frac{\pi}{2}\right)}$$

Figure 3.9: Single pulse Logic gate - NOT

$$\boxed{\text{H}} = \boxed{iR_x\left(\frac{\pi}{2}\right)} \boxed{R_z(\pi)}$$

Figure 3.10: Two pulses Logic gate - Hadamard

$$\begin{aligned} R_x(\theta) &= e^{-i\theta X/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \\ R_y(\theta) &= e^{-i\theta Y/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} \\ R_z(\theta) &= e^{-i\theta Z/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \end{aligned} \quad (3.3)$$

- Two-qubit operation; depending on approach the Interaction operator is used as I_{zz} or I_{xy} for various rotations θ

Thus a quantum circuit realized in NMR will be exclusively built from single qubit rotations about three axes x, y, z and from the two-neighbor-qubit operation of interaction allowing to realize such primitives as CNOT or SWAP gates. Examples of gates realized using NMR quantum primitives are shown in Figure 3.9 to Figure 3.12.

Also, the synthesis using the NMR computing model using EM pulses,

Figure 3.11: Detailed Realization of Feynman gate with five EM pulses.

Figure 3.12: Five pulses Logic gate - Controlled-V

is common to other technologies such as Ion Trap [CZ95, PW02] or Josephson Junction [BZ00]. Thus the cost model used here can be applied to synthesize circuits in various technologies, all of these technologies having the possibility to express the implemented logic as a sequence of EM pulses.

3.6 Principles of Synthesis for NMR technology

3.6.1 V - Gate, T - Gate and the principle of "Level-generalization"

The principle of creating permutative circuits using the V and the $V^\dagger$ gates can be extended to SWAP gate. This is possible because SWAP gate has a NOT³ sub-matrix as can be seen in equations 3.4 and 3.5. Equation 3.4 represents the SWAP gate and its Square-root, the T gate. The equation 3.5 represents the nSWAP (nS) gate and its Square-root.

³ Feynman, Fredkin and Toffoli gates are all a combination of the Control signal and the NOT target bit. Thus they do all have a NOT sub-matrix in their complete unitary matrices

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow \sqrt{S} = T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1+i}{2} & \frac{i-i}{2} & 0 \\ 0 & \frac{1-i}{2} & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.4)$$

$$nS = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightarrow \sqrt{nS} = nT = \begin{pmatrix} \frac{1+i}{2} & 0 & 0 & \frac{1-i}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1-i}{2} & 0 & 0 & \frac{1+i}{2} \end{pmatrix} \quad (3.5)$$

The nS (invented by me and introduced in [LP05b]) gate function is the opposite to the SWAP gate: while the SWAP gate exchanges the values of the qubits when they differ ($|01\rangle$ and $|10\rangle$) the nS gate swaps values when both qubits are equal ($|00\rangle$ and $|11\rangle$).

Both of these gates have the V sub-matrix and thus the T gate is a good candidate for logic synthesis similarly to the powerful *CV*. Again, the right side of equation 3.4 reveals the V core of the T gate. Compared to *CV* gate, the matrix is shifted diagonally left-up. For the nT gate (square-root-of-nSWAP gate) the V core can be seen dispersed to the four corners of the matrix.

Figure 3.13: SWAP gate broken into two T gates

Figure 3.14: Fredkin gate broken into two Controlled-T (CT) gates

$$\text{and } T \times T = S, \quad nT \times nT = nS, \quad T \times nT = nT \times T = I \quad (3.6)$$

The function realized by the Controlled-Swap (CS) is a conditional swapping of two target qubits. For instance let $|\phi\rangle = \frac{|110\rangle + |101\rangle}{\sqrt{2}}$ be an initial state and applying CS yields $CS|\phi\rangle \rightarrow \frac{|101\rangle + |110\rangle}{\sqrt{2}}$. The unitary matrix of the T (square-root-of-swap) is very similar to the matrix of the CV and thus can be expected to have similar powerful properties when used in synthesis. Thus, replacing the CV gates in the Peres gate by the CT gate gives a possible lead to explore such scaling properties (Figure 3.15). This is true for all gates from the Peres family (see below) and thus also for all families from Figure 3.16. For convenience this property of replacing gates with larger gates but preserving the relative structure of the circuit will be called the 'level-generalization'.

Figure 3.15: The level-generalization of Peres using CT . From a three-qubit circuit with (CV and $CNOT$ gates) at the left, the four qubit circuit (using $CNOT$ and CS) is related as the "Level-generalization".

3.6.2 Insertion principle

Similar to the previous property for synthesis, the generalization of the concept of gate insertion is introduced. We will call our new principle the Peres-Toffoli-Fredkin (PTF) principle. The PTF principle just states that based on the simple transform required to go from Peres to Toffoli and Feynman, there can be more similar simple insertion or removal operations giving similar relation between different 'interesting' circuits. Using this PTF principle the exploration of the quantum circuit problem space is well situated to techniques exploring local groups (blocks) of gates and circuits. This is because the PTF principle can be simulated and exhaustively searched using simple exhaustive search. The PTF principle is shown in Figure 3.16. By adding more Feynman gates, several interesting circuits such as the Miller gate [LPMP02] can be directly created.

3.6.3 The Divide and Conquer Principle

The last property introduced here is the principle of "divide and conquer", explained on the following example. A single Fredkin gate is broken down

Figure 3.16: Peres, Toffoli and Fredkin gates illustrating simple search technique (PTF principle).

into two *Controlled* – *T* gates and by simple adding of Feynman gates many new circuits are created. This example is illustrated in Figure 3.17. This new gate generation principle results from combining and generalizing two previous principles: the level-generalization and the PTF principle.

Figure 3.17: The "divide and conquer" circuit synthesis method

The above simple and powerful principles and tools to construct quantum circuits are sufficient to describe the low level of the GAEX (Chapter 6), the evolutionary-exhaustive-transformative software 'explorer' for the quantum circuit synthesis. This program is explained in the sequel.

3.6.4 The Gate-collapsing principle

The last important principle in the QLS is the so called gate-collapsing principle. It is based on the fact that as each quantum gate is represented by a

unitary matrix and any neighboring gates that are on the same qubits can be collapsed into a single one. This is shown in Figure 3.18. The requirements for the gate collapsing principle are the following:

- gates must be neighbors (adjacent)
- gates must be defined on the same qubits
- gates must have same width

Figure 3.18: The gate-collapsing principle: quantum gates located on the same qubits and having no other gates in between can be *collapsed* into a single one.

The gate-collapsing principle can be also applied in specific cases when some of the above conditions are not fulfilled. Figure 3.19 shows two such cases (c) and (d). In order to collapse two non adjacent gates the following conditions must be fulfilled:

- moving gates left (or right) cannot change the value on any other control qubit (Figure 3.19a and 3.19b)

- moving gates left (or right) can be done if it affects only target qubit (Figure 3.19c and 3.19d)

Figure 3.19: The gate-collapsing principle: quantum gates can also be collapsed if they can be moved and become neighbors without altering the controls of other quantum gates.

3.7 Examples of circuits obtained automatically for NMR technology using methods from sections 3.5 and 3.6

Now let us continue the discussion of the circuit decomposition into primitives started earlier in this chapter. Let us consider one practical example. The Toffoli or Fredkin gates introduced in section 3.5 are both universal quantum logic gates that are already well-known. They have been built in several quantum and reversible technologies. The problem that I will be solving here is to find an optimal decomposition of the universal gates into smaller parts, especially the directly realizable quantum primitives such as Feynman, NOT or Controlled-V

(CV) gates. As mentioned earlier, the gates with one and two qubits have costs directly dependent on the number of EM impulses. Thus using the result from [LLK⁺06], the individual costs for every single gate are the following: $W = 1$, $Phase = 1$, $H = 2$, $CNOT, CV = 5$, $Swap = 11$, $Peres = 12$, $Toffoli = 13$, $Fredkin = 19$. Figure 3.20 presents the well-known realization of Toffoli gate from [SD96]. There are five 2-qubit primitives here: CV_{23} , CV_{13} , $CNOT_{12}$, $CV_{23}^\dagger$, $CNOT_{12}$, and the cost is $5 * \text{Cost}(CNOT) = 25$. The subscript on each gate name signifies the wires that the gate is connected to. For instance on a three qubit circuit the gate $CV_{23}^\dagger$ is controlled by the second qubit and applies the conditional $V^\dagger$ transformation to the third qubit.

With respect to a synthesis method and the definition 3.5.6 of QLS, the circuit implementing Toffoli gate with the above cost is the ideal solution (as using this set of quantum gates there is no better (less expensive) realization of this function). Thus a minimization of a circuit cost with respect to a known minimum will allow to both find circuits directly reducible to the ideal one or circuits different than the ideal circuit. This will be seen later (chapter 5) where we show circuits realizing the Toffoli gate with the same cost and same component gates as well as circuits realizing the Toffoli gate with a higher cost and with the realized function being the correct one. Observe that the presented transformations in these examples are all minimizing the cost of quantum circuits. Interestingly, these cost reducing methods allow to transform a circuit form being built using one set of quantum gates to another gate using a different set of quantum gates. Thus the cost minimization does not

Figure 3.20: Toffoli circuit from 2×2 quantum primitives

Figure 3.21: Toffoli-based Fredkin circuit from Feynman gates and a macro-Toffoli gate

alter only the circuit structure but creates also new quantum gate primitives that can be used for QLS.

Using the PTF principle (Figure 3.16), we can realize the Fredkin gate from the Toffoli gate. Using a GA [WG98, LPG⁺03] or the synthesis methods given earlier in this dissertation, we can synthesize the Fredkin gate using two Feynman gates and one Toffoli gate as in Figure 3.21. The cost of this gate is $2 \cdot 5 + 25 = 35$.

Substituting the Toffoli design from Figure 3.20 to Figure 3.21 we obtain the circuit from Figure 3.4a (top). Now we can apply an obvious EXOR-based transformation to transform this circuit to the circuit from Figure 3.4b (middle). This is done by shifting the last gate at right (Feynman with EXOR up) by one gate to left. The reader can verify that this transformation did not change logic functions realized by any of the outputs. Observe that a cascade

of two 2×2 gates is another 2×2 gate, so by combining a Feynman with EXOR-up gate (cost of 5), followed by controlled-V gate (cost of 5) we obtain a new gate CV with the cost of 5. Similarly gate $CV^\dagger$ with cost 5 is created (the unitary matrices of both CV and $CV^\dagger$ are shown in equation 3.7).

$$\begin{aligned}
 CV &= [NOTC] \times [CV] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 1 & 0 & 0 \end{pmatrix} \\
 CV^\dagger &= [NOTC] \times [CV^\dagger] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 1 & 0 & 0 \end{pmatrix}
 \end{aligned} \tag{3.7}$$

This way, a circuit from Figure 3.4c (bottom) is obtained with the cost of 25. (This transformation is based on the method from [LLK⁺06] and the details of cost calculation of CV and $CV^\dagger$ are not necessary here). Thus, the cost of Toffoli gate is exactly the same as the cost of Fredkin gate, and not half of it, as was previously assumed and as may be suggested by classical binary equations of such gates.

Encouraged with the above observation, that sequences of gates on the same quantum wires have the cost of only single gate on these wires, we used the same method to calculate costs of other well-known gates. Let us now investi-

Figure 3.22: Stages of the minimization of the Miller Gate. Observe that by using the Peres macro and the collapsing principle the overall cost of the gate is reduced. The important fact is that the collapsed blocks C_V and $C_{V^\dagger}$ can be much less expensive in some technologies than in others.

gate a function of three majorities investigated first by Miller [Mil02, MD03, YHSP03]. This gate is described by equations: $P = ab \oplus ac \oplus bc$, $Q = \bar{a}b \oplus \bar{a}c \oplus bc$, $R = a\bar{b} \oplus ac \oplus \bar{b}c$. Where $\bar{a}$ (a' in Figure 3.22) is a negation of variable a . Function P is a standard majority and Q , R are majorities on negated input arguments a and b , respectively [YHSP03]. We realized this function with quantum primitives, found it useful in other designs and thus worthy to be a stand-alone 3×3 quantum gate. We call it the Miller gate [YHSP03] and we found a solution that is less expensive than that of prof. Miller.

Our realization of the Miller gate requires 4 Feynman gates and a Toffoli gate [LPG⁺03] (Figure 3.22a), which would suggest a cost of $4 \cdot 5 + 25 = 45$. Performing transformations as in Figure 3.22b, we obtain a solution with cost 35. Another solution obtained by the same method has cost 35 and is shown

Figure 3.23: Example of comparison of synthesis using Toffoli and Peres primitives.

in Figure 3.22c. It is also based on simple EXOR transformation $(x \oplus y) \oplus z = (x \oplus z) \oplus y$ applied to three rightmost Feynman gates from Figure 3.22a, with EXOR in the middle wire y . Again, the Miller gate, based on its binary logic equations, looks initially much more complicated than the Toffoli gate, but a closer inspection using quantum logic primitives proves that it is just slightly more expensive. Observe that these minimization methods are NMR technology related, as for instance in the quantum dot reversible technology with no ancilla bits the Miller gate is the least expensive gate. Also, remark that similar rules and software can be used for other quantum technologies when the basic gates are known.

Finally observe that the main reason for searching for novel quantum primitives

is the minimization of large quantum circuits. For instance Figure 3.23 shows the difference of cost while building a larger circuit with Toffoli and with Peres type primitives. Observe that when built with Toffoli primitives the resulting cost is 12×2 quantum gates while using the Peres primitives the cost is only 8×2 quantum gates.

3.8 Chapter Conclusion

In this Chapter I presented the principles and methodologies used in Quantum Logic Synthesis. In the following chapters, the knowledge from this and from chapter 2 is explored from an algorithmic manner. It would be possible to explore each presented method manually but in general an algorithm automating these methods is preferred so as it can be tested on real life problems. For this reason in the next chapter I introduce a specially designed genetic algorithm that is used in this dissertation as the automated quantum logic synthesizer.

4 GENETIC ALGORITHM FOR LOGIC SYNTHESIS

4.1 Introduction

A Genetic Algorithm (GA) is an evolutionary model of search and learning based on observed and simplified principles from natural phenomena; the evolution of species in Nature. It was originally proposed by Holland [Hol75] but it was later extended to various alternatives allowing the evolutionary approach to formulate and possibly solve problems from various domains [Hol75, Gol89, MT78, LPMP02, Rai96, Yen05]. The whole research area of evolutionary computation can be separated into three main subareas: Genetic Programming (GP), Evolutionary Strategies (ES) and Genetic Algorithms. The main differences reside in the problem representation (GP - represents the problem as logical trees, GA as a string of bits, ES - string of real numbers), in the employed evolutionary operators (Mutation, Cross-Over, Adaptive Mutation) and finally in the way of generating and selecting solutions.

From the computational point of view, the class of GA's is well suited to solve the constraint-satisfaction problems (CSP) because it offers certain advantages in problems with a high number of local maxima and in a noisy problem definition. Some instances of the CSP problems that GA was applied to are the Traveling-Salesman Problem, the Eight-Queen problem, the Satisfiability problem, the Map Coloring, etc. Also, quantum logic synthesis is a CSP prob-

lem when viewed as a method to synthesize a desired function as a quantum circuit given some structural constraints such as the width of the circuit (number of qubits), the possible types of gates to be used or the total number of gates to use.

The approach described by the evolutionary computation represents a metaphor to the evolutionary process in Nature described at best by a natural selection, sexual reproduction and random mutation seen as a process of computation.

First, these concepts mean that a GA uses chromosomes (strings of elements) to represent the problem; each chromosome can be a possible solution to the problem. For a population of such *individuals*, a large problem space can be covered by the evolutionary search as well as multiple local minima can be explored. Second, a population of individuals together with the fitness function represents the information about the problem that is available to the evolutionary algorithm. Third, the computation is represented by a computational cycle that consists of: random mutations (introducing noise and novelty into the system - randomly altering the individuals), information exchange between individuals (local solutions) represented in the cross-over operation and a simulated *survival-of-the-fittest* mechanism of selection and replication. These are the only operations allowed on the set of individuals. Fourth, the method of evaluation of each individual contains all of the knowledge required to determine whether or not a solution was found.

This brief description can be illustrated by the following points describing the

3	2	1	7	5	8	4	6	9	0
---	---	---	---	---	---	---	---	---	---

Figure 4.1: Example of an individual in a GA solving the TSP problem.

process of designing a GA for a given problem.

- Assuming the problem is formalized as a CSP, select the appropriate information encoding. For instance, a n-vertices TSP problem can be encoded as strings of length n, representing by an integer number each vertex from the graph. This can be seen in Figure 4.1
- Select appropriate evolutionary operators, in general the standard mutation modified for the problem is enough to introduce noise, however self-adaptive mutation operator is also a well known tool in ES approach. In particular for the TSP problem one must preserve the chromosomes such that each chromosome represents a valid path in a graph (each vertex must be present only once). Thus the mutation operation can be a 2-vertices random position swap or inversion.
- Select a cross-over operator such as single-point cross-over, two-points cross-over or multi-parent crossover [EvKK95]. For certain problems, when the chromosome is ordered, the crossover must be able to preserve the overall validity of the individual similarly as in the case of mutation. Again, in the case of TSP, Figure 4.2 explains the crossover.
- Select or create a fitness function - the function that will evaluate each individual solution and generate fitness value for each solution. This

Figure 4.2: Example of cross-over operation for the TSP problem.

means that for the TSP problem, such a function evaluates the total path length that each individual's chromosome encodes for.

- Select a selection scheme such as Roulette Wheel (RW) or Stochastic Universal Sampling (SUS). This process simulates the *survival-of-the-fittest* rule, as it is biased to select preferably chromosomes/individuals with a higher value of fitness. In this step it is also possible to tune the selection process to various forms of elitism (These concepts are explained in details below).

In this chapter a GA is presented and its mechanisms are explained in details. The various functions and objects are specified with respect to the problem of quantum logic synthesis (synthesis of quantum circuits). More general ideas are also introduced below in order to cover the potentials of the evolutionary computation in quantum circuit (automata, etc) design. The description also extends to the mechanism of the combined approach called the GAEX genetic algorithm for quantum circuit synthesis [LPG⁺03, LP02, LP05b].

4.2 Genetic algorithm

A Genetic algorithm is a set of directed random processes that make probabilistic decisions - simulated evolution. Table 4.1 shows the general structure of a GA algorithm and this section follows this structure with each step explained in individual sub-section.

Table 4.1: Structure of a Genetic Algorithm

```

01:  $t \leftarrow 0$ ;
02: initialize( $P(t)$ );                                /* initial population */
03: while (not termination-condition) do
04:   evaluate( $P(t)$ );                                /* evaluate fitness */
05:    $t \leftarrow t + 1$ ;
06:    $Q_s(t) \leftarrow select(P(t \leftarrow 1))$ ;      /* selection operator */
07:    $Q_r(t) \leftarrow recombine(Q_s(t))$ ;            /* crossover operator */
08:    $P(t) \leftarrow mutate(Q_r(t))$ ;                /* mutation operator */
09: end while

```

4.2.1 Encoding/Representation

For quantum logic synthesis the representation that we use is based on the encoding introduced in [LPMP02]. This representation allows to describe any Quantum or Reversible circuit [LPG⁺03, LP02]. All individuals in the GA are strings of ordered characters (each character representing a quantum gate) partitioned into parallel blocks. This partitioning of the circuit was in our case induced from the representation of any QC such as one in Figure 4.3. Each block has as many inputs and outputs as the width of the quantum array (five in the case of Figure 4.3). The chromosome of each individual is

4.2.2 Initialization steps of GA

The GA requires an input file (c.f. Pseudo-Code 4.1, 4.2 and Pseudo-Code 4.3) which specifies all input parameters and required settings.

```
01 :Population Size:80
02 :Segment MaxNumber:20
03 :Segment MinNumber:2
04 :Local Search generational switch:100000
05 :Mutation Probability:10
06 :Crossover Probability:85
07 :Alpha:0.99
08 :Beta:0.01
09 :Cost divider:10
10 :Type of GA (normal=0; Baldwinian=1) : 0
11 :Type of Mutation(normal=0; bitwise=1) : 1
12 :Type of Crossover(1-point = 0;2-point=1) : 0
13 :Replication(0 – RW, 1 – SUS, 2 – Tournament) : 1
```

(4.1)

```

14 :Type of Fitness(0 – simple linear, 1 – simple square,
    2 – complex linear, 3 - complex) :0
15 :Type of fitness calculation (0 - individual, 1 - grouped) :0
16 :Pareto optimization (0 - no, 1 - yes) :0
17 :Threshold replication (0 - no, other - threshold) :0
18 :Elitism (0-disabled, 1-enabled) :0
19 :Tournament parameter(amount of individuals chosen randomly) :5
20 :Measurement :1
21 :Measured qubits indexes:0
22 :(1,0)(0,1)(1,0)(0,0)(1,0)(0,0)(0,0)(0,0)
23 :(0,0)(0,1)(0,0)(1,0)(0,0)(1,0)(1,0)(1,0)

```

(4.2)

The lines (01-23) within the file specifies the parameters defining the overall behavior of the GA. The number of individuals in the population is given first in line 01. The size (length) of the circuit specified by a maximal (t_{max}) and minimal number of segments (t_{min}) in each individual (chromosome) is given in lines 02 and 03 respectively.

The initial circuits are created with a random size within the interval specified by these two parameters. The size of the chromosome is not limited during the

lifetime of an individual. Rather, two other parameters allow to specify the minimal and maximal sizes of every circuit. Each individual has a dynamically changing genome and the GA is a subclass of the Messy GA [GKD89].

Line 04 specifies the number of generations of the evolutionary search after which the GA will switch to a local search. Lines 05-06 specify the probability of mutation and crossover, and lines 07 and 08 specify the parameters α and β when fitness function from eq. 4.17 and 4.18 is used. Line 09 represents the minimal cost $MinCost^1$ used in the cost function described in section 4.4.2.

Line 10 specifies the type of GA that is run; two possibilities are available a standard GA and a Baldwinian model, line 11 specifies whether bitwise mutation or standard mutation is used, line 12 determines if single-point or two-point crossover is used and line 13 allows to select the replication mechanism: either the Stochastic Universal Sampling (SUS), the Roulette Wheel (RW) or the Tournament Selection can be used. Line 14 specifies the type of the fitness function (Section 4.4) and the line 15 allows to share the fitness among individuals (fitness scaling). Line 16 allows to turn the GA into a Pareto-optimizing (multi-objective) evolutionary search and line 17 allows to force the GA to limit the selection and replication process by a threshold. Line 18 allows to use the Elitism and line 19 allows to choose the number of individuals used in the tournament replication procedure. Finally lines 20 to 23 specify if the measurement is used, how many qubits are measured as well

¹ In this dissertation this parameter will be referred to as either *minimal* cost, *desired* cost or *optimal* cost during the process of synthesis

as the expectation values of the measurement.

$$\begin{aligned}
&23 : 21 \\
&24 : 1 \\
&25 : 1 \\
&26 : \text{wire} \\
&27 : (1, 0)(0, 0) \\
&28 : (0, 0)(1, 0) \\
&\quad \vdots \\
&44 : 3 \\
&45 : 1 \\
&46 : \text{Controlled_wire_V} \\
&46 : (1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
&47 : (0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
&48 : (0, 0)(0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
&49 : (0, 0)(0, 0)(0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
&50 : (0, 0)(0, 0)(0, 0)(0, 0)(0.5, 0.5)(0.5, -0.5)(0, 0)(0, 0) \\
&51 : (0, 0)(0, 0)(0, 0)(0, 0)(0.5, -0.5)(0.5, 0.5)(0, 0)(0, 0) \\
&52 : (0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0.5, 0.5)(0.5, -0.5) \\
&53 : (0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0.5, -0.5)(0.5, 0.5)
\end{aligned} \tag{4.3}$$

The input file also lists the elementary quantum gates to be used as components, like the single qubit H, X, Y, Z or V gates and two qubit operations such as *CNOT* or *CV*, which are the building blocks of the quantum circuits to be found. The quantum gates are represented as quantum unitary (and Hermitian) matrices with the cost specified for each gate.

On line 23 the total number of component input gates is given. Then the unitary matrices of the gates, their number of in/output and their cost are given. From lines 23 to 27 the truth table of the single qubit operation "Wire" is presented. Another quantum gate is depicted as a quantum truth table (in a form of a unitary matrix) from lines 44 to 53. This gate is a 3*3 Controlled-V (CV) gate [BBC⁺95] on qubits at index 0 and 2. Observe that each input gate is specified by a unitary matrix that describes each complex coefficient by the real and the imaginary component. For instance (1,0) represents the real state while (0.5,0.5) represents a complex state with coefficient $\frac{1+i}{2}$.

The above described features of the GA are described later on in this chapter.

4.3 Evaluation of Synthesis Errors

The GA created by me has two possible evaluation methods for designed circuits that have been developed in order to accommodate both completely and incompletely specified quantum-reversible functions: ME and EE.

4.3.1 Element Error Evaluation method (EE)

To represent a completely specified functions (in particular, the deterministic permutative reversible functions such as the universal gates Fredkin or Toffoli) a matrix is used. It can be seen that as long as the function is easily specified in this manner, it is possible to evaluate the Unitary matrix of the synthesized circuit directly. This matrix represents the desired matrix that the circuit must satisfy by comparison of each matrix coefficient. For instance, to specify the Fredkin gate, the number of the qubits of the result and the matrix specifying the target circuit can be added at the end of the input file (eq. 4.4). Observe that each coefficient in the matrix is represented by a pair of floats. For instance (1,0) represents the number $1 + 0i$ and (0,0) represents $0 + 0i$.

$$\begin{aligned}
54 : & 3 \\
55 : & (1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
56 : & (0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
57 : & (0, 0)(0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
58 : & (0, 0)(0, 0)(0, 0)(1, 0)(0, 0)(0, 0)(0, 0)(0, 0) \\
59 : & (0, 0)(0, 0)(0, 0)(0, 0)(1, 0)(0, 0)(0, 0)(0, 0) \\
60 : & (0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(1, 0)(0, 0) \\
61 : & (0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(1, 0)(0, 0)(0, 0) \\
62 : & (0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(0, 0)(1, 0)
\end{aligned} \tag{4.4}$$

This method is computationally intensive. However, it allows to directly alter and analyze the circuit matrix representation; changes on individual gates can be directly observed on the unitary matrix. The error evaluation is done in this case by $e_k = \sum_j (o_j - s_j)^2$ (by comparing the resulting and the desired matrices coefficient-by-coefficient) in order to obtain the overall error (before measurement). This evaluation method is referred to as the Elements Error (EE) method.

4.3.2 Measurement Evaluation method (ME)

When the desired circuit is to represent an incompletely specified function represented as $f = [00, --, --, 10]$ or as $f = [0-, -1, --, 0-]$, the matrix representation is not convenient and designing a unitary quantum-realizable incompletely specified function might not even be possible for large functions. Also, representing such circuit as a unitary matrix would require to specify all elements either as cares or as don't cares. In general, the number of don't cares for machine learning will be much higher than the number of cares. Thus representing it as a matrix would be wasteful.

Thus, in cases when the function is incompletely specified (or the output function is defined on less bits than the input), it is better to represent the solution only by the obtainable information. This is done by using the after-measurement evaluation of the circuit. The GA generates the set of measurement operators for each desired qubit. For each output state of the circuit

under evaluation the algorithm measures the state for the desired combined measured state. In the case of a don't care the GA skips the given measurement and the value of the output remains unknown. This method is referred to as the Measurement Evaluation (ME) method.

4.3.2.1 Measurement Evaluation Input-Data Specification

The specification of the problem for ME method is shown on lines 19 to 22 (eq. 4.2). In this case, there is one qubit that is going to be measured (line 19) on the first wire (indexed as 0: line 20). The measurement expectation values are in lines 21 for the expected state $|0\rangle$ and line 22 for the expected state $|1\rangle$. Note that the measurement for state $|001\rangle$ (line 21 and line 22) has expected complex value $0 + i1$ represented as $(0, 1)$ for both possible outcomes $|0\rangle$ and $|1\rangle$. This artificial notation means that the outcome is considered as a don't care. The error evaluation becomes $e_k = \sum_j (o_j - s_j), \forall j \in O$, with O the set of all defined expected values.

For example, assume the output state from the circuit under evaluation is

$$|\psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \text{ and the expected result is the state } |00\rangle. \text{ The error with respect}$$

to the measured qubit (assume indexed at 0) will now be $e_0 = 0$ and for the second qubit $e_1 = 0.5$. This means that if the output is taking into account only the zero-th qubit, the state $|\psi\rangle$ is a valid solution. However, if also the

Table 4.2: Example of the Majority gate encoding for the GA. Observe that each input minterm with qubit $|q_3\rangle = |1\rangle$ is a don't care.

$q_0q_1q_2q_3$		m_0	m_1	$q_0q_1q_2q_3$		m_0	m_1
0000	$\xrightarrow{M}$	(1,0)	(0,0)	1000	$\xrightarrow{M}$	(1,0)	(0,0)
0001	$\xrightarrow{M}$	(0,1)	(0,1)	1001	$\xrightarrow{M}$	(0,1)	(0,1)
0010	$\xrightarrow{M}$	(1,0)	(0,0)	1010	$\xrightarrow{M}$	(0,0)	(1,0)
0011	$\xrightarrow{M}$	(0,1)	(0,1)	1011	$\xrightarrow{M}$	(0,1)	(0,1)
0100	$\xrightarrow{M}$	(1,0)	(0,0)	1100	$\xrightarrow{M}$	(0,0)	(1,0)
0101	$\xrightarrow{M}$	(0,1)	(0,1)	1101	$\xrightarrow{M}$	(0,1)	(0,1)
0110	$\xrightarrow{M}$	(0,0)	(1,0)	1110	$\xrightarrow{M}$	(0,0)	(1,0)
0111	$\xrightarrow{M}$	(0,1)	(0,1)	1111	$\xrightarrow{M}$	(0,1)	(0,1)

second qubit is used, the state $|\psi\rangle$ is not a solution, because the expectation value for the output state $|00\rangle$ is 0.5. In general, the error is summed over all measured qubits and normalized in order to be directly used in the calculation of the fitness function. The particular function, shown in lines (21-23), is the majority (3x1) function with the measured output on the 0-th qubit. To synthesize circuits using an ancilla bit constant such as in [HSY⁺06], the input is modified to represent the whole information after measurement with using don't cares for all values of the ancilla bit/constant that are not required. For example to represent the majority function as a three-input and single-output function (using a dedicated output qubit q_0 set to $|0\rangle$), the measurement is specified in Table 4.2. This table describes the function from the K-map in Figure 4.4a.

The topmost line shows the input state $q_3q_2q_1q_0$ in the first column, columns labeled m_0 and m_1 represent the encoded probability of observing the state

Figure 4.4: Majority gate function: (a) the K-map, and (b) the schematic circuit representation.

$|0\rangle\langle 0|$ and $|1\rangle\langle 1|$ respectively (single qubit output specified by a pair of numbers $(1, 0)(0, 0)$ representing the complex coefficients for the state $|0\rangle$ and $|1\rangle$). The $\xrightarrow{M}$ operation represents the whole state being measured using projective measurement on orthonormal bases $|0\rangle$ and $|1\rangle$. The don't care is represented as an complex expectation value $(0, 1) (0, 1)$. The corresponding K-map with the general circuit structure is shown in Figure 4.4a,b.

4.3.3 Comparison of EE and ME

The error for each individual is calculated by comparing each user-specified output care with the value obtained from the simulation of the quantum circuit behavior.

If the ME model is used, the desired outputs are specified as the state of the

system after the measurement. The desired output is defined by the state that has the highest probability of observation for a given input state. For permutative circuit this means that for each input state the GA will be evaluating the probability of observing two states: the desired and the undesired state. For instance, if the desired output state is $|001\rangle$, the GA will also evaluate the probability of observing the most undesirable state to $|001\rangle$ which is $|110\rangle$.

The state $|110\rangle$ is called the undesired state and represents the negation of the desired state. This means that a given function to be synthesized as a circuit is specified by the user using a set of desired states. The undesired states are used to provide additional information during the evaluation and the evolutionary search for the solution circuit.

Each desired output is calculated for each measured value individually. For a two valued output, the error with respect to the state after measurement of 0 and 1, can be written as $\frac{(o_k(0)-p)^2+(o_k(1)-(1-p)^2)}{2}$, with p being the desired probability of obtaining a 0 and $q = 1 - p$ is the probability of obtaining a 1. For a complete set of inputs the overall error for a given individual is given by

$$\begin{aligned} error &= \sum_k e_k = \frac{1}{k} \sum_{k=1}^{2^n-1} \sum_{j=0}^{m-1} (o_k - \langle \psi'_k | M_j^* M_j | \psi'_k \rangle)^2 \\ &= \frac{1}{k} \sum_{k=1}^{2^n-1} \sum_{j=0}^{m-1} (o_k - p_k)^2 \end{aligned} \quad (4.5)$$

with $|\psi'_k\rangle = U|\psi_k\rangle$, m being the number of possible outcomes of the measurement, $\langle \psi | M_0^* M_0 | \psi \rangle = p$ and $\langle \psi | M_1^* M_1 | \psi \rangle = 1 - p$.

For a $\frac{1}{2}$ -spin quantum system (Boolean observable), $j = 2$, the equation 4.5 can be rewritten as

$$error = \frac{1}{k} \sum_k \frac{(o_k(0) - p_k)^2 + (o_k(1) - (1 - p_k)^2)}{2} \quad (4.6)$$

Thus for an incompletely specified permutative function defined on three qubits, the measurement is designed as $M_J = m_{01}^\otimes n$ and represents the Kronecker product of single qubit measurements on j qubits. Each single qubit measurement is selected according to the preference given by the GA software user. For instance the desired two-qubit measured state given as $(1,0)(0,0)(0,0)(1,0) \rightarrow$

$$|10\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \text{ will be measured using the measurement operator from eq. 4.7}$$

$$M_2 = m_1^1 \otimes m_0^0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (4.7)$$

A further constraint on ME evaluation method is that for each function the number of measured qubits must remain constant. This requirement insures that each measurement result has the same weight in the overall error (error is directly proportional to the number of measured qubits).

Observe that using the ME error is much less precise than using the EE error. For a three qubit function, the matrix based method compares all 64 coefficients of the matrix while the ME evaluates only 16 output states. The implication is that the error obtained in ME has less information about the system.

When the desired state is deterministic (reversible permutative function), the correct output state can be obtained exactly up to the phase of the unmeasured state. In the case, when the desired single-qubit state observation probability is not deterministic, both states are detected by the single-qubit measurement operator. For instance eq 4.8 shows how to interpret the encoding of a single qubit superposed quantum state.

$$\begin{aligned}
 (0.5, 0)(0.5, 0)(0, 0)(1, 0)(0, 0)(1, 0) &\rightarrow \frac{1}{\sqrt{2}}(|000\rangle \pm |100\rangle) \\
 &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)|00\rangle
 \end{aligned} \tag{4.8}$$

The first qubit's is described by $(0.5, 0)(0.5, 0)$, the second and the third qubits are described by $(0, 0)(1, 0)$. Thus the state can be factored with respect to the first qubit as shown in eq 4.8.

For multi qubit measurement the probability of a desired output state is the product of probabilities of observation for all individual qubits; various multi-qubit states observed under two measurements of maximally opposed observables can be indistinguishable.

The GA always builds only two measurement operators. From the external observer point of view, this measurement process loses information proportionally to the number of simultaneously measured qubits. For instance the desired states in eq. 4.9 and 4.10 are not distinguishable under this incomplete measurement.

$$\begin{aligned}
 (0.33, 0)(0.69, 0)(0.1, 0)(0.9, 0)(0, 0)(1, 0) &\rightarrow \frac{1}{30}|000\rangle + \frac{2}{3}|110\rangle \\
 &\quad + \frac{1}{3}|010\rangle + \frac{2}{30}|100\rangle \quad (4.9) \\
 &\rightarrow (\frac{1}{30}|00\rangle + \frac{2}{3}|11\rangle)|0\rangle
 \end{aligned}$$

$$\begin{aligned}
 (0.1, 0)(0.9, 0)(0.33, 0)(0.69, 0)(0, 0)(1, 0) &\rightarrow \frac{1}{30}|000\rangle + \frac{2}{3}|110\rangle \\
 &\quad + \frac{2}{30}|010\rangle + \frac{1}{3}|100\rangle \quad (4.10) \\
 &\rightarrow (\frac{1}{30}|00\rangle + \frac{2}{3}|11\rangle)|0\rangle
 \end{aligned}$$

Observe that the desired state is $|110\rangle$ (with observation probability $0.69 \cdot 0.9 \approx \frac{2}{3}$) and the undesired state is $|000\rangle$ (with observation probability $0.1 \cdot 0.33 \approx \frac{1}{30}$) and because two of the measured qubits are in superposed state half of the possible observables are lost. The lost information are the possible output states $|100\rangle$ and $|010\rangle$. Thus the encoded state in eq. 4.10 is not a complete quantum state but the described encoding is sufficient to encode problems studied in this dissertation.

The measurement of multi-qubit entangled states is also possible by specifying equal probabilities of observation for each possible output state (eq. 4.11). In

such a case, the desired and undesired state are such that the qubits in each state have the same value.

$$(0.5, 0)(0.5, 0)(0.5, 0)(0.5, 0)(0, 0)(1, 0) \rightarrow \frac{1}{4}|000\rangle + \frac{1}{4}|110\rangle \quad (4.11)$$

The purpose of using two different evaluation methodologies was to allow more control over the evolutionary search. In general the measurement evaluation is used for the exploration; searching for novel functions or for incompletely specified function synthesis. The EE approach is more useful to describe completely specified permutative functions for the cost and the circuit size optimization.

Observe that because the EE method is a difference of squares between the outputs of the target and the synthesized quantum circuit, the obtained unitary matrix of a circuit can differ from the target circuit by phase or by a imaginary component i . This is illustrated later on in section 5.4.3. Also, because when using the ME method the GA generates two measurements for each possible output, the algorithm can search for the target circuit or for the negation of the target circuit. The synthesis of negated circuits is interesting because the desired circuit can be easily generated from such circuit simply by negating all qubits. Synthesis of a negated circuit is illustrated in section 5.3.2.

4.4 Fitness functions of the GA

The error defining the correctness of a (potential) solution is used in the fitness function that allows to shape the landscape of the solutions in order to find the global optimum. The fitness function quantifies how good the individuals (candidate solutions) are. As already mentioned, the fitness function is the mechanism allowing to determine the correctness of a given individual. In order for the fitness function f to correctly approximate the problem space:

- $\forall c_i \in G, \exists f(c_i)$, i.e. f must be continuous
- $f(c_i) \geq 0$, i.e. f must exist for every individual problem representation.

The fitness function evaluates, during each generation, all the individuals of a population and it determines which individuals are more likely to "survive" and which will be possibly discarded. The fitness value of each individual represents how close the individual is to the optimal solution represented by the fitness value 1.

The evaluation of each individual is based on the fitness function. It requires sometimes various degrees of adjustment of the fitness value. One of such parameterizations is the penalty function. The penalty function represents a negative component (lowering the fitness value of individuals) in cases where an individual c_i gets outside of the validity of its phenotype - the individual hit the boundary constraints. Thus the general form of an evaluation function

$e(c)$ is shown in eq. 4.12.

$$e(c) = f(c) + p(c) \quad (4.12)$$

with $f(c)$ being the fitness function and $p(c)$ is the penalty function. A simple penalty function is equal to zero for valid individuals ($p(S) = 0, \forall S \in F$) and strongly non zero for invalid individuals. There are several variants of penalty functions e.g. Death Penalty, Static Penalties, Dynamic Penalties, Annealing Penalties, Adaptive Penalties, Segregated GA and Co-evolutionary Penalties [Yen05]. The penalty function is not directly used in the GA but a cost function is introduced in order to (a) manipulate the general shape of the fitness function and (b) to explore the relation between the correctness of quantum circuits and their cost.

4.4.1 Simple Fitness Functions

Four different fitness functions are implemented and can be chosen by declaring them in the input file.

$$f_1 = 1 - \frac{Error}{E_m} = 1 - e \quad (4.13)$$

The first function in eq. 4.13 is the simplest and represents fitness inversely dependent on the overall error. The maximal error E_m is calculated through

$$E_m = 2^{2^n} \quad (4.14)$$

with n being the number of wires/qubits. This error can be normalized using the equation 4.6.

The second fitness function is described by equation 4.15.

$$f_2 = \frac{1}{Error + 1} \quad (4.15)$$

The fitness function f_2 preserves a small probability for the less fit individuals due to its exponential character. It allows individuals with very small fitness to be selected during replication with a higher probability than fitness f_1 . These two fitness functions are graphically represented by the bold lines in Figure 4.5. The overall error is calculated for an example of a 3-qubit circuit (the maximal error is $2^{2^3} = 64$).

4.4.2 Cost Based Fitness Functions

The cost function is based on a parameter known as the *minimum cost* that is provided by the user and that permits to estimate a normalization constant. This means that the cost function acts as a bonus inversely proportional to the size of the circuit to the fitness function for a given estimated and unreachable minimum. In this dissertation the *cost function* is defined by

$$G(c) = \exp^{-\frac{(MinCost - Cost)^2}{\frac{Cost^2}{2}}} \quad (4.16)$$

where *Mincost* is the parameter given by the user (line 09 eq. 4.1)

Figure 4.5: Schematic representation of the four available fitness functions. The solid lines represent respectively the fitness functions f_1 and f_2 and the dashed lines represent the fitness functions f_3 and f_4 .

and $Cost$, given by $\sum_{j=1}^k c_j$, is the sum of costs of all gates in the evolved circuit. Equation 4.16 was experimentally determined to be sensitive enough to influence both circuits far and close to the optimal cost.

The two weighted fitness functions (eq. 4.17, 4.18) calculate the fitness value using the fitness function and the cost function together. In this case, the fitness is dependent on the error of the gate with respect to the truth table of the specification as well as on the cost of the circuit. Each component of these weighted functions can be adjusted by the values of parameters Alpha and Beta (line 6+7 in pseudo-code 4.1). The two weighted fitness functions

are given in equations 4.17 and 4.18, respectively.

$$f_3 = \alpha (1 - e) + \beta G(c) \quad (4.17)$$

$$f_4 = \alpha \left(\frac{1}{Error + 1} \right) + \beta G(c) \quad (4.18)$$

In Figure 4.5 the dotted lines which represent the weighted fitness functions are shown assuming that the cost function is constant for all the error values. The second term from eq. 4.17 and 4.18 - the cost - was set to -0.05 to represent the worst case for settings $\alpha = 0.95$ and $\beta = 0.05$. This corresponds to a circuit that is so long that for the user-defined minimum its cost is very large, thus its fitness is penalized by $f(s) = 0.95 * E + 0.05 * 0 = E - 0.05$.

In other words, in the two weighted fitness functions from eq. 4.17 and 4.18, the term $\beta * Cost$ lowers the fitness functions 3 and 4 by a constant number in comparison with the fitness functions 1 and 2. This formulation of a weighted fitness function allows the selection process to explore solutions not only related to the correctness of the circuit itself but also to explore the problem space in a different cost-based neighborhood of solutions. This is because each individual would be affected in a similar manner and the overall length of the individual strings allows the GA to overcome local minima such as $f_{\alpha_i, \beta_i}(s_k) \geq f_{\alpha_j, \beta_j}(s_l), e_k < e_l$. On one hand the α and β coefficients allow individuals with lower circuit cost and higher error to reproduce and on the other hand they allow to progressively adjust the overall cost of the pool of individuals.

The reasons for these various fitness functions are the following:

- to allow different selection pressures during the individual selection process this will be discussed in Section 4.5,
- by calibrating the cost to always underestimate the minimal possible size of the desired circuit it is possible to further manipulate the selection process.

For instance the fitness function 4.17 is not equal to one, unless both the cost of the circuit and the error are minimal. Thus a GA using such a weighted function has more freedom for searching a solution, because the fitness function is now optimizing the circuit for two parameters. Similarly in the case of the fitness function 4.18 this type of fitness function lowers the fitness function values of longer circuits, therefore preferring the shorter ones. Thus individuals with different circuit properties will have equal fitness value. Let two individuals s_0 and s_1 have the fitness calculated according to eq. 4.19 and eq. 4.20.

$$f(s_0) = \alpha E_0 + \beta C_0 \quad (4.19)$$

$$f(s_1) = \alpha E_1 + \beta C_1 \quad (4.20)$$

Then it is obvious that $f(s_0) = f(s_1)$ for $E_0 < E_1$ and $C_0 > C_1$ such that for $\Delta C = C_0 - C_1$, $\Delta E = E_1 - E_0$ we can write $E_0 = E_1 - \Delta C$. This expression represents the fact that two circuits, one with larger error but shorter in size than the other one will have more chances to get selected for replication. The

method thus preserves the diversity of the population to a larger extent. As will be seen later, this property is an important requirement for successfully synthesizing certain types of quantum circuits.

The general expression for the cost function is given by eq. 4.21,

$$C_{c_i} = \frac{C_{Min}}{Cost_i} \quad (4.21)$$

where $Cost_i \geq C_{Min} > 0$ and $Cost_i$ is the cost of the given solution (circuit). This function requires that the user given minimum (C_{Min}) is not realizable for the given circuit with cost $Cost_i$.

In the case where the estimated minimum is not well known (a smaller circuit implementing desired function might exist), an exponential Cost function can be used, as in eq. 4.22.

$$C_{c_i} = e^{-|(C_{Min}-Cost_i)|} \quad (4.22)$$

This exponential cost function is less sensitive to variations in size of larger circuits and at the same time gives smaller circuits a distinct cost advantage. Thus this cost function is well situated to be used on hard problems where a premature convergence occurs often.

4.5 The Selection Process

The process of selection chooses the best individuals for reproduction. Two (or more) parent individuals from the current population are chosen for the reproduction. The selection process simulates the principle of the natural selection by preferring the more fit individuals to the less fit ones. Thus individuals with higher value of fitness are selected more often (with a higher probability) than those with lower fitness values. With each individual being a potential solution or carrying a piece of its genotype required to find the solution, an appropriate selection method of individuals must be applied to find the problem solution. This means that for a successful search one requires that the selection pressure is low at the beginning of the search and the selection pressure becomes high towards the end of the search. The idea behind the *selection pressure* is the following: from an initial random population of individuals, the solution will be found if the selection process preserves enough of variety in the genetic pool of the population to allow overcome the local fitness maxima. This means that if the selection picks only individuals with the highest fitness (*the high selection pressure case*), the global solution might not be found because the individuals do not have enough genetic material to generate the solution. On the other hand, if the selection is too relaxed (*the low selection pressure case*) the search process will take too long and might not converge at all. The selection of individuals will become independent of their fitness value which reduces the evolutionary process to a random search.

In general, the initial population generation is very important to the success of the evolutionary computation. This is because during a GA computation the information that is contained in the population of the individuals decreases with every generation as the main computational operation is the individual recombination or crossover. As will be seen later, the replication and crossover do not bring any new information into the genetic pool. The mutation operator does bring external (random) changes in the population, but in general it is used only to perturb the system, rather than insert large amounts of random data. A mutation process inserting too much random elements will again reduce the evolutionary process to a random search. In this case the selection process will pick individuals proportionally to their fitness value, but each generation being highly modified by the mutation operator does not allow any predictable convergence to the global optimum solution.

This GA uses the fitness proportional selection (eq. 4.23). This approach is mainly known from the Roulette Wheel [Bak87, Gol89] and the Stochastic Universal Sampling methods[Bak87]. The Roulette Wheel is the simplest of the selection methods, it allows to randomly select two individuals with probabilities of selection proportional to the fitness values of these individuals. This is shown in equation 4.23.

$$P(c_i) = \frac{f(c_i)}{\sum_{j=1}^n f(c_j)} \text{ with } \sum_{j=1}^n f(c_j) > 0 \quad (4.23)$$

$p(c_i)$ is the probability with which an individual is chosen at the selection

Figure 4.6: The fitness normalization represented on a line (a roulette wheel). Each individual has a segment allocated proportional to its fitness

Figure 4.7: The Roulette Wheel Selection - a random number generator picks the individuals one by one located on the equi-distant points over the individuals fitness-sum representation. Here, $p \approx 0.45$ selects individual c_4 .

with n being the population size. Every individual is represented by a section of a circle proportional to its $p(c_i)$ and follows the normal probability rules (eq. 4.24). This is shown in Figure 4.6. Thus the whole population can be represented as a roulette wheel with sections proportional to $p(c_i)$.

$$\sum_{i=1}^n p_s(c_i) = 1 \quad (4.24)$$

The "ball" inside the roulette wheel represents a random number within the interval $[0, 1]$. The individual, which section is "hit" by the roulette ball is selected (Figure 4.7). Here it is c_4 .

The Stochastic Universal Sampling (SUS) [Bak87] is similar to the well known Roulette Wheel selection method. Every individual obtains again a section on the roulette wheel proportional in size to the fitness value. A fixed number of individuals $l \geq 2$ is chosen, this is the number of individuals that a single

step of the SUS selection procedure will select for recombination (in Figure 4.8 it is c_4). Now create another ruler with l equi-distant points on it and of the same length as the roulette wheel. Finally a random number is generated in the interval $[0, \frac{1}{l}]$ that indicates the location of the first point on the ruler on the roulette wheel. This can be equally obtained, by wrapping the ends of the roulette wheel (to create a circle) and then generating a random number in the interval $[0, 1]$ and allocating the remaining $l - 1$ points on the roulette wheel accordingly.

Figure 4.8: For N given individuals to select, pick a random number, and from its location pick $N-1$ equidistant points on the fitness landscape. In this case the selected individuals are c_2, c_4, c_7 and c_{10} .

In our implementation two individuals are selected for replication when using the SUS selection. Therefore a random number in the interval $[0, 1]$ is produced and multiplied by the fitness sum of all individuals. Since only 2 individuals should be combined, the second individual is determined by adding $\frac{1}{2} \bmod 1$ to the first generated random number and the result is multiplied by the fitness sum of all individuals.

Another selection mechanism is the Tournament Selection, that allows to overcome some disadvantages of the fitness proportional selection methods. In this

case to select an individual c , one chooses randomly k individuals (uniformly distributed) from the population and takes the best individual (one with the highest fitness). The selection pressure is in this case controlled by the parameter k . If k is increased, the selection pressure will be raised as well. The probability of an individual of being selected using this method is shown in equation 4.25.

$$p_s(c_i) = \left(1 - \left(1 - \frac{1}{n}\right)^k\right) \times \left(1 - \frac{i}{n}\right)^{k-1} \quad (4.25)$$

The Tournament Selection is the last selection method that is implemented within my GA for QLS. The number of individuals which are randomly chosen can be controlled via a parameter in the input file. In Table 4.2 line 16 the number of the tournament participants was set to 5.

4.6 Crossover and Mutation

The crossover is the primary operator of a GA. With the crossover, new individuals are produced out of selected parents by exchanging information (piece of their genome) between parents and creating new recombined offsprings. The crossover is applied to all individuals with a probability $p > 0$. There are various types of crossover operators, [Rai96].

The simplest crossover method is the **single-point crossover**. After the selection, a random number is generated such that $p_0 = \text{rand}(\text{length}(c_1))$, $p_0 < \text{length}(c_1)$, $p_0 < \text{length}(c_2)$, with $\text{length}(c_i)$ is the length of the genome of the

Figure 4.9: single-point crossover

i^{th} individual. This random number marks the position where the crossover should take place for both individuals. This is shown in Figure 4.9 for the encoding introduced in Section 4.2.1. The single-point crossover can also be implemented. A random location of crossover is selected for each of the individuals participating in the exchange. This is shown in Figure 4.9.

The **two-point crossover** is the second crossover method. At the beginning, two random numbers are created for each individual selected for the recombination process. These numbers select the two positions within the first individual where the recombination is applied. Both numbers may not exceed the length of the first individual. The same is done for the second individual. This is illustrated in Figure 4.10.

The **Uniform Crossover** is a generalization of the n-point strategy such that for every genome it is decided by random choice from which parent-individual

Figure 4.10: Two point crossover: in both parents, two random points are selected and the segments between such points from each parent are exchanged.

it is taken. A random number on the unit interval is generated for each gene. If $r < 0.5$ the gene is taken from the first parent individual, otherwise it is taken from the second parent individual.

4.6.1 Mutation

After the Evaluation, Selection and Crossover a mutation is applied with a given probability (Table 4.1 in line 3). If a random number is below the mutation probability, then a mutation is applied. The mutation is a secondary operator and serves to insert new or lost gene material into the population, in general with a very small probability $p < 0.1$. If mutation is performed too often, the search process degenerates to a complete random search. If the mutation is applied too rarely it does not create jumps large enough in the genomes and thus does not help to overcome some local maxima (Section 4.5).

Various mutation operators are available[Rai96], here we will describe some of them. The simplest mutation operator is a single bit flip (a random change) represented using our encoding in Figure 4.11a. This operation is naturally extended to a bitwise mutation, where the mutation operator is applied on each bit (gate) in the genome. By using the Swap Mutation (Fig. 4.11b), also called Exchange-Based Mutation, the contents of the random selected genes is exchanged (Figure 4.11a).

Figure 4.11: *Flip* mutation a) bitwise *Flip* mutation b) and *Swap* mutation c)

In a more detailed mutation variant, the mutation operator can be separated into three distinct actions on the genome. First, a random number selects a position within the individual and the chosen segment will be replaced (Fig. 4.12).

The second mutation performs an erasure of one segment somewhere within the individual on a given random position (Fig. 4.13).

The last mutation type adds one segment to the end of an individual (Fig. 4.14).

Figure 4.12: Replacement of a segment - the first mutation method

Figure 4.13: Erasure of a segment - the second mutation method

The ratio of how often each of these mutation types are applied in the GA is a distribution of choices over a uniform random distribution. This is because the fact that the mutation operator is applied to the full-width circuit blocks. For example, for a circuit that is built as $[H] \otimes [CNOT] * [W] \otimes [X] \otimes [Z]$ assume that the $[X]$ gate was selected for mutation. If the replacement gate is of the same size (in this case one qubit), the gate is just replaced, and the circuit remains otherwise unchanged. In the case when the dimension of the replacement gate is greater than the size of the original gate, the whole segment containing $[W] \otimes [X] \otimes [Z]$ is deleted and regenerated so as to contain the replacement gate. Similarly, in the case when the gate selected as the

Figure 4.14: Adding of a segment - third mutation method

replacement has less qubits than the original gate, the segment is completed by empty strings so that the width of the segment remains constant.

4.6.2 Additional GA tuning strategies

4.6.2.1 Replacement Strategies

The replacement strategy represents the method by which old individuals (parents) are replaced by the new ones (offsprings). There are various approaches to the replacement; the most common are: *generational GA*, *Steady-State GA* and *Elitistic GA* [Gol89].

4.6.2.2 Generational GA

In the generational GA, the population of offspring individuals completely replaces the population of their parents. Thus all information *transmitted* from generation to generation is completely contained in the offsprings and all information contained in the parent generation is discarded. This approach can generate such an offspring population that the best individual of the offspring generation is worse with respect to the best parent individual.

4.6.2.3 Elitism

To solve the problem of losing information just by the selection problem the Elitism strategy is one of the best known to choose. The Elitism allows to preserve the best individuals from the parent population to be saved in the children population. Thus no matter the results of the selection process or the average fitness, the n-best individuals from the parent population are copied into the offspring population. When using Elitism, the GA copies only one best individual from the parent population to the new generation in order not to lose the best individual during the replication process.

4.6.2.4 Steady-State GA

Another approach to the problem of premature convergence, is the so-called *Steady-State GA*. Similarly to Elitism, the Steady State GA also preserves some individuals from the parent population and copies them to the offspring population; however proportions in the overall mechanism are inverted. This time most of the population is kept unchanged and only a small number of individuals is changed. The Steady State GA evaluates both populations (parent and offspring) together, and only the best individuals from both the parent and offspring populations are taken into the next generation. Unlike Elitism, however, the Steady State GA is based on the principle of overlapping population and the fact that for finding the solution only small and more controlled steps can be used. Genetic algorithms, which are based on this strategy, tend

to converge faster. Higher mutation and crossover probabilities can be used with the Steady-State-GA, because good population members are protected through the selected replacement strategy (replacement of individuals with lower fitness values).

4.6.2.5 Boundary-Constraints

Another type of general restrictions imposed on the GA are the Boundary constraints. These constraints can be simple bounds (specification of minimum and maximum) or a linear coupling of parameter values (max or min of a function defined over the set of all elements of the individual encoding). They can be related to the feasibility or to the quality of the solution. A genetic algorithm achieves the best solution, if the encoding, the initialization and the operators of a problem are chosen, such that all possible individuals are valid solutions. That means that all boundary constraints are always satisfied and thus no mechanism needs to be considered to repair the potentially invalid individuals. However, it is possible that such an integrity of the genome cannot be assured and thus illegal genomes may be created. Therefore, these invalid individuals should get a worse evaluation, so that they won't be reproduced in the next generation.

4.6.2.6 Repair Mechanism

Beside avoiding the violation of the Boundary constraints by designing such a GA that does not generate invalid offsprings, a repair mechanism can be used. In our approach, all generated circuits are valid circuits. However the recombination can generate individuals that have very long genomes and mutation can generate completely (temporarily) invalid individuals. Individuals that are too long are not desired because they are out of the initial specifications and thus a repair mechanism is able to shorten the genome. Individuals that undergo mutation as in Figure 4.12 can be modified so that the circuit represented by them would be invalid. As can be seen on the top-right of the Figure 4.12, the genome is mutated within one parallel segment: $pISWIp \rightarrow pIEWp$. However, if only mutation would be applied the result would be $pISWIp \rightarrow pIEWIp$, which is an invalid circuit. Thus a repair mechanism has been implemented so that the final chromosomes in every population are valid circuits. It is shown schematically in Figure 4.15.

4.6.2.7 Termination Condition

The term *termination condition* represents a set of rules that will terminate the evolutionary computation. In the simplest case, this condition terminates the computation if one of the following happens:

- A pre-specified maximum number of iterations (generations) was reached (absolute termination),

Figure 4.15: Example of the repair process. The repair mechanism is able to break down individual segments of gates and reconstruct them so as the circuit remains structurally valid.

- an optimal or a sub-optimal solution was found.

In our GA the termination condition is reached if one of the individuals of the population has no error with respect to the specified gate. The information about this search including the used quantum gates with all information (truth table etc.) is printed, as well as the result with the truth table, the cost and the genotype of the individual. Also, since there is no guarantee to find a solution, the GA terminates after 10000 generations.

4.7 Chapter Conclusion

The presented algorithm was designed for the exploration of quantum circuits and quantum computing. In the next chapter the main experimentation is presented on a set of selected benchmarks that outline most appropriately the main goal of this dissertation: the experimental and algorithmic exploration of

quantum circuit synthesis. These benchmarks include some of the well known universal reversible quantum gates such as Fredkin or Toffoli as well as multiple qubit entanglement circuits.

5 EVOLUTIONARY SEARCH FOR LOGIC SYNTHESIS

In this chapter we present some of the the experimental settings, results, extensions and observations of the initial evolutionary approach to Quantum Logic Synthesis. In particular, this chapter describes how by using the GA (Chapter 4) we were able to re-synthesize already known universal gates and minimize them in some cases.

As introduced in Chapter 3 we will realize all gates from 1×1 (called also 1-qubit gates) and 2×2 (i.e. 2-qubit gates) quantum realizable component gates. Although our methods are aimed to discover a logic circuit representing a desired function, in order to be more technology specific we assume in some variants of the search the use of quantum gates representing the Nuclear Magnetic Resonance (NMR) quantum computers [YSPH05, ZLSD02, LLK⁺06]. Also, as shown in previous chapters (Chapter 3 and 4), we will assume that the cost of every gate is calculated using a selected quantum cost for each experiment. Finally the overall goal of this Chapter was to demonstrate the methodology used to find the least costly (in number of gates) realization of well-known Toffoli, Fredkin, Miller, Peres and Peres family gates as well as to design new quantum gates given their complete specifications.

The experimentation covers benchmarks that can be separated into the following sub-categories:

-
- Exact Synthesis problems of completely specified functions
 - Evolutionary synthesis of permutative universal gates such as Fredkin, Toffoli, Majority and Miller gate using exact measurement evaluation (single and multi-qubit measurement)(ME evaluation Section 5.3)
 - Evolutionary synthesis of permutative universal gates such as Fredkin, Toffoli and the Entanglement quantum gates using exact matrix evaluation (EE evaluation Section 5.4)
 - Approximate Synthesis problems of incompletely specified functions
 - Evolutionary Synthesis of permutative universal gates such as Fredkin, Toffoli and some additional benchmark functions (Section 5.3, 5.4 and Chapter 7)
 - Algorithmic, user-driven pseudo-random exhaustive search for permutative circuit structure-based quantum logic synthesis using the GAEX and EX algorithm.(Chapter 6)
 - Approximate Synthesis problems of incompletely specified functions for Machine Learning (Chapter 7)
 - The synthesis of quantum circuits with quantum output states for novel robotic behaviors.(Chapter 7)
 - The synthesis of FSM represented as quantum circuits (Chapter 7)

This Chapter is organized as follows. Section 5.1 presents the framework of the presented experiments, section 5.2 discusses the experimental settings and conditions of the GA. Section 5.3 discusses the obtained results of the evolutionary search using the ME evaluation methodology and section 5.4 analyze the results of the evolutionary search using the EE evaluation. Finally section 5.5 concludes this chapter and discusses all benchmarks and results used in this dissertation.

5.1 *Evolutionary Search*

The synthesis of quantum logic circuits using evolutionary approaches has two fundamental aspects:

- First, it is necessary to find the circuit that either (A) exactly corresponds to the specification, or (B) differs only slightly from the specification. Case (A) is verified by a tautology of the specification function and the solution function. In case of a truly quantum circuit this is done by a comparison of unitary matrices. In case of permutation functions this can be also done by comparing the truth tables. Observe that non-permutative matrices cannot be represented by truth tables which leaves the representation of unitary matrices as the only canonical function representation. This representation is responsible for less efficient tautology verification during fitness function calculations, which considerably slows down the software execution time. Case (B) calculations

for permutation circuits are verified by an incomplete tautology (tautology with accuracy to all combinations of input values and with arbitrary logic values for don't care combinations). In some applications such as robot control or Machine Learning it is sufficient that the specification and the solution are close, like, for instance, differing only in a small percent of input value combinations (Chapter 7).

- Second fundamental aspect of quantum logic synthesis is that the cost of the circuit has to be as close as possible to the known minimum, in order to allow the least expensive possible quantum hardware implementation (like the minimum number of electromagnetic pulses in NMR technology).

5.2 *Experimental setup of GA*

5.2.1 *Input Sets of Quantum Primitives*

Chapter 3 presented a general approach to the calculation of the cost using single or/and two-qubit quantum gates/pulses as unit components of the total circuit cost. For higher level gates (such as CCNOT, Fredkin, etc.), we assume that each gate has cost equal to the sum of its component gates in a given model. The single qubit gates have a unit cost 1 (including Identity). The two qubit operations have the cost either given along with the definition of the gate (by the user) or they must be realized by the Algorithm from smaller primitives. In such a case the cost of the used universal gates is equivalent to

the realization that the algorithm has built; all gates are either derived from the initial gate set or are in the initial set.

As introduced in Chapter 3, the gates used were separated into sets. The experiments described in this chapter were using three sets. Each input-gate set contains a small set of single qubit unitary transformations in generally selected from the set $S_1 = \{\text{Wire}, X(R_x(\pi)), Y(iR_y(\pi)), Z(iR_z(\pi))\}$.

The three input gate sets categories are:

1. Limited angle rotations: $S_{lr} = S_1 \cup \{R_x(\theta), R_y(\theta), R_z(\theta), I_{zz}(\theta)\}$ (with angles $\theta = \pm\pi, \pm\frac{\pi}{2}, \pm\frac{\pi}{3}$),
2. Full: $S_f = S_1 \cup \{\text{H}, \text{CNOT}, \text{CCNOT (Toffoli)}, \text{SWAP}, \text{C_SWAP (Fredkin)}, \text{Majority}\}$.
3. Partial: $S_p = S_1 \cup \{\text{SWAP}, \text{V}, \text{V}^\dagger.\text{C_V}, \text{C_V}\}$,

The S_{lr} group represents one of the most general units of quantum computing - single qubit rotations and the interaction gate. All four operators in S_{lr} are parameterized by θ and thus the set size depends on the desired and allowed precision. This input-gate set was used to verify results from [LLK⁺06] and search the underlying problem space. However, because each logic operation in a system built from these smallest segments (NOT is a single pulse, Hadamard two pulses, CNOT five pulses, CV five pulses, etc) it is very difficult for the GA to keep so many small component gates together and thus the synthesis using this set is one of the most difficult.

The group S_f represents the full set of gates (using universal gates as input) and was used to determine the functionality of the GA. In particular, the full set was used to verify the GA capacity to re-synthesize a gate. Naturally when the target gate is an element of the full set, the target gate is removed from the set of input gates. This set is also used when searching for more difficult permutative functions.

The third set S_p is one of possible partial sets used to search for smaller universal gates and for novel realizations of universal gates. In general, the set S_p contains a small subset of quantum gates. For instance most common partial input-gate sets are $\{I, H, CNOT\}$, $\{I, V, V^\dagger, CV, CV^\dagger, CNOT\}$, $\{I, X, Y, Z, Phase, CNOT\}$ and so on.

The results are separated into two categories based on the ME and the EE error evaluation methods. In both cases, the results of synthesis of Toffoli, Fredkin and Miller are analyzed in details for both ME and EE evaluations. Additional benchmarks are provided in an overall discussion of the synthesis results describing the performance of the GA.

Table 5.1: Fixed parameters during search for Toffoli gate

Population	100	Generations	200
Mutation	0.05	Crossover	0.8
t_{min}	7	t_{max}	12
σ	6		

5.3 Discussion of the Results of the Evolutionary Quantum Logic Synthesis using ME evaluation

5.3.1 Toffoli Gate

The Toffoli gate was successfully reinvented by our GA as well as novel implementations have been found. In this section, the GA performance is analyzed with the focus on the multi-qubit measurement and the single qubit measurement. The experiments are presented and analyzed with respect to fixed parameters shown in Table 5.1.

The Table 5.1 shows parameters t_{min} and t_{max} (representing the approximate size limits of the circuits analyzed) that limits the problem space to circuits between seven and twelve segments and with a user-given minimal cost 6. This cost is suboptimal, despite the fact that as already shown, the Toffoli gate can be synthesized using the S_p set of gates with 5 two-qubit gates. Thus the default cost is 5, but because even applying Identity on a qubit requires the cost equivalent to any other single qubit unitary operation, the minimal cost of the Toffoli gate is $C_{toffoli} = 5 + 4 = 9$; there are four single qubit Identities in a Toffoli gate.

The reason that all gates (including Identity) have to have a cost, is the fact that if there were gates with cost 0, the synthesizer would prefer cheap gates over other gates with higher cost that could be useful in the synthesis process. Consequently gates with no cost could lead to erroneous circuits creating a local minimum leading the evolutionary process to less successful searches (the GA is stuck in some local optimal fitness). This means that circuits with a high error and low cost will have the same fitness as circuits with lower error and higher cost.

Figures 5.1 and 5.2 represent the averages over 50 runs of the fitness value, the current best fitness value, the cost and the average error per generation as well as the fitness, the error and the cost for the currently best solution.

5.3.1.1 Single-qubit ME model

Figure 5.1 shows the results of the evolutionary synthesis using single-qubit measurement (measuring only single qubit) and Figure 5.2 illustrates the multi-qubit measurement (measuring all output qubits). Figures 5.1a and 5.1b represent the results of the search for Toffoli gate using single qubit ME and using between 10 and 20 and between 20 and 40 segments respectively.

Observe that the single qubit measurement is statistically successful when the GA is exploring the problem subspace of a circuit size where the solutions can be found quickly. This can be seen on Figure 5.1a: cost decrease over the evolution of the GA and the overall fitness increases indicating the algorithm

Figure 5.1: Results for Toffoli gate using the single-qubit measurement for two different settings of t_{min} and t_{max} . Observe the overall convergence of the evolutionary search in (a) and compare to relative non-convergence in (b) due to a over-estimated parameters t_{min} and t_{max}

convergence. Such convergence, is due to the fact that in the allotted time, most of the 50 test GA runs are already terminated and thus providing the maximal fitness and minimal error to the statistical representation.

However, such convergence is only observed when the parameterization of the GA corresponds to an easily found global minimum. Such parameterization is generally unknown but can be either determined

- experimentally - by combining parameters and observing results from performed experiments
- theoretically - by specifying parameters closest to a known minimal realization of a gate.

Observe that in Figure 5.1b the size of the circuit given by t_{min} and t_{max} is too large and the evolutionary process is less successful.

5.3.1.2 Multi-qubit ME model

Figures 5.2a and 5.2b represent the results of searching for Toffoli gate using multi-qubit ME for fitness function calculation and using between 10 and 20 segments and between 20 and 40 circuit segments respectively. In all these experiments the fitness function from eq. 4.15 is used.

When using the multi-qubit measurement, the output is a 3×3 reversible function and as can be seen the evolutionary process is less successful (Figure 5.2a and 5.2b). This is due to the fact that in the allotted number of generation

(a) Three-qubit measurement $t_{min} = 10, t_{max} = 20$ (b) Three-qubit measurement $t_{min} = 20, t_{max} = 40$

Figure 5.2: Results for Toffoli gate using the multiple-qubit measurement for two different settings of t_{min} and t_{max}

cycles, only an approximate solution was found. This effect is observed for most of the experiments, but as is shown below, when the synthesis process is reduced to a single qubit function, the circuit might not be an exact match.

Figures 5.2a and 5.2b can be commented by the following observations.

- The first set of curves represents the averages of the population (fitness value, the current best fitness value, the cost and the average error). This set of curves also shows the trend of the evolutionary process. The shape of these curves is in most graphs slowly increasing as more and more solutions are found. Observe that the average values can be almost constant when the solutions are rare.
- The second group (the fitness, the error and the cost for the currently best solution) represents the average fitness of the best individual and the bottom line represents the associated error for the best individual. This set is well observable; the fitness is the topmost curve mapped on the primary axis. The error has a shape mirrored along the line $fitness = 0.5$ and the cost curve has the same shape as when using the disproportionate fitness (mapped on the secondary axis)
- Observe that the convergence of the GA is not visible, despite that the algorithm found correct solutions (Example of this can be seen in Figure 5.3 where a successful search for Toffoli gate is shown despite the overall non-convergence of the evolutionary search). The lower convergence rate is due to the fact that while synthesizing with multiple single qubit

measurements, the solution must match three disjoint observations. Naturally this is more difficult than when using a single measurement. Also, similarly to the single-qubit measurement case, increasing the parameters t_{min} and t_{max} generates worse results if these parameters overestimate the size of possible solution circuits. Finally, the main differences between Figures 5.2a and 5.2b are the circuit cost (observe that in Figure 5.2b the cost of circuits is much larger as a consequence of synthesizing larger circuits) and the circuit fitness (in Figure 5.2b the fitness is lower, again as a result of the overestimation of the circuit size).

5.3.1.3 Comparison of Single-qubit and Multi-qubit ME methods

Figure 5.4 shows several Toffoli gates that were discovered using our GA using the single-qubit ME error model. Observe, that despite all these circuits represent correctly the single output function, the other qubits in the whole state can be still in a superposition; for instance eq. 5.1 shows the unitary matrix of the circuit in Figure 5.4c representing the single-output Toffoli function, that

Figure 5.3: Example of successful synthesis of the Toffoli gate using a partial input gate-set S_p and a single-qubit ME method.

is built from controlled V and controlled- $V^\dagger$ gates only.

$$\begin{pmatrix}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0.5 - 0.5i & 0 & 0 & 0 & 0.5 + 0.5i & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 0.5 + 0.5i & 0 & 0 & 0.5 - 0.5i & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0.5 + 0.5i & 0 & 0 & 0 & 0.5 - 0.5i & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0.5 - 0.5i & 0 & 0 & 0.5 + 0.5i & 0
 \end{pmatrix} \quad (5.1)$$

Figure 5.4: Circuits discovered by the GA as a single output Toffoli Gate using the single-qubit ME evaluation. (Unitary matrix representing it is given in eq. 5.1)

Observe that for any three qubit logic input, the operator from eq. 5.1 generates a 3×3 quantum probabilistic function; the whole output state $U|\psi\rangle$ can be in a superposition (eq. 5.2),

$$\begin{aligned}
 U|001\rangle &\rightarrow \frac{1+i}{2}(-i|001\rangle + |101\rangle) \\
 &\xrightarrow{\text{Measurement}} \frac{1}{2}|001\rangle\langle 001| + \frac{1}{2}|101\rangle\langle 101|
 \end{aligned} \tag{5.2}$$

while the measurement of only the output qubit generates deterministic output corresponding to the Toffoli gate. Both of the possible outputs are shown in Table 5.2.

The main point of this section was to demonstrate that using single-qubit measurement for quantum circuit synthesis, a $n \times 1$ reversible logic function can be easily synthesized but at the cost of losing the permutative reversibility on other qubits. In this case, the Toffoli gate was synthesized using the single-

Table 5.2: Example of a 3×3 quantum probabilistic and a 3×1 deterministic functions generated by the circuit from Figure 5.4c.

$q_2q_1q_0$	All Measured $q_2q_1q_0$	Single Measure q_0
000	$ 000\rangle\langle 000 $	$ 0\rangle\langle 0 $
001	$\frac{1}{2} 001\rangle\langle 001 + \frac{1}{2} 101\rangle\langle 101 $	$ 1\rangle\langle 1 $
010	$ 110\rangle\langle 110 $	$ 0\rangle\langle 0 $
011	$\frac{1}{2} 011\rangle\langle 011 + \frac{1}{2} 111\rangle\langle 111 $	$ 1\rangle\langle 1 $
100	$ 100\rangle\langle 100 $	$ 0\rangle\langle 0 $
101	$\frac{1}{2} 001\rangle\langle 001 + \frac{1}{2} 101\rangle\langle 101 $	$ 1\rangle\langle 1 $
110	$\frac{1}{2} 011\rangle\langle 011 + \frac{1}{2} 111\rangle\langle 111 $	$ 1\rangle\langle 1 $
111	$ 010\rangle\langle 010 $	$ 0\rangle\langle 0 $

qubit ME, which generates the single output reversible logic function on the output target but the control qubits are either not restored to the original input values or can even remain in some quantum superposed or entangled state (Table 5.2).

5.3.2 Synthesis of Fredkin Gate

Table 5.3: Fixed parameters during search for Fredkin gate

Population	100	Generations	200
Mutation	0.05	Crossover	0.8
t_{min}	10	t_{max}	20
σ	10		

Similarly to Toffoli, the Fredkin gate was successfully re-synthesized and novel circuit implementations have been found, using our GA. Similarly to the case of the Toffoli gate, the results of the experiments are reported with respect to fixed parameters.

5.3.2.1 Single-qubit ME model

synthesis of such *negated* gate is possible as described in section 4.3.3.

Figure 5.6: Example of realization (with minimization) of Fredkin gate using two-qubit ME model and the S_f input-gate set. Observe that all presented gates contain the well known realization of Fredkin ([*NOTC*] [*TOFFOLI*] [*NOTC*]) on the qubits b and c , but with additional gates changing the value on the third, unmeasured qubit a .

Because the Fredkin gate is essentially a single-controlled-qubit two-qubit-target unitary gate, the ME method can be specified for two measured qubits. Some of the results using the full input-gate set S_f are shown in the Figure 5.6. As can be seen the synthesized gates can be minimized to the well known Fredkin realization but as only two qubits are measured, the third qubit is modified by an additional gate.

5.3.2.2 Analysis of using Partial vs. Full input-gate set.

In this section, the GA performance is analyzed with the focus on the effect that the size of the input gate-set has on the convergence of the algorithm. Table 5.3 shows the parameters that have been used for the following experiments as constants. In this case, t_{min} and t_{max} parameters are increased in order to give the GA enough space for the search. The minimal cost σ remains the same, and as in the Toffoli case, it underestimates the cost of a minimal Fredkin gate built using the GA.

The synthesis process was more difficult (smaller success rate and longer runs required) in the case when incomplete sets of gate have been used. Figures 5.7 and 5.8 represent two sets of runs, both using fitness function from eq. 4.15 and ME evaluation.

Figures 5.7a and 5.7b represent the search experiments using the S_f input gate-set and Figures 5.8a and 5.8b shows the results from the same experimental settings but using the S_p input gate-set.

Observe, that Figures 5.7a and 5.7b illustrate the problem of over-estimation of the size of the circuits with respect to the given input set. Figures 5.8a and 5.8b illustrate the problem of under-estimation of the size parameters: observe that in Figure 5.8a the GA is stuck in a local minimum (using the S_p input gate-set for $(t_{min}, t_{max}) = 3, 6$ no solution is found; the best individual fitness average is stuck at local minimum as the lower upper bound t_{max} does not allow the GA to explore larger circuits.), while in Figures 5.8b the algorithm

(a) Full set S_f , $t_{min} = 3$, $t_{max} = 6$ (b) Full set S_f , $t_{min} = 13$, $t_{max} = 23$

Figure 5.7: Comparison of results for Fredkin gate using the full input gate-set S_f for two different settings of t_{min} and t_{max} . Similarly to the case of Toffoli gate, the overestimation of the size parameters leads to poorer results.

(a) Partial set S_p , $t_{min} = 3$, $t_{max} = 6$ (b) Partial set S_p , $t_{min} = 13$, $t_{max} = 23$

Figure 5.8: Comparison of results for Fredkin gate using the partial input gate-set S_p and for two different settings of t_{max} and t_{min} . (a) - the value of parameters t_{max} and t_{min} is too small for the GA to find a solution and thus the statistical average is stuck (shown as a straight line on the top) in a locally best solution. (b) - an increase in size allows the GA to explore larger circuits and also to find solutions, despite the fact that overall convergence is not visible.

searches larger circuits allowing it to find solutions.

Example of successful search for Fredkin gate is shown in Figure 5.9 illustrating as in the case of Toffoli gate that the evolutionary process finds a solution despite not converging. Observe, that during the evolutionary run, the GA fitness (in each generation) is stuck at 0.8 and when the solution is found fitness goes to 1 and error goes to 0 in a single jump.

Figure 5.9: Example of successful synthesis of the Fredkin gate using a full input gate-set S_f and a two-qubit ME method.

These observations confirm that the algorithm behaves according to our expectations; by increasing the size of the input set the time to find the solution increases as well. This is natural as the GA can chose from more input com-

ponent gates and thus the search space is larger. Moreover these results show the fact that using a universal gate in the input-gate set naturally reduces the required size of the circuits to be built than when such a gate is missing from the input-gate set. This is because as the GA is an artificial-evolution driven random process, circuits may carry a lot of unminimized gates. Thus to create more efficiently such gates, a logic minimizer or heuristic methods can be added to provide the desired boost.

5.3.3 Synthesis of Majority Gate

Figure 5.10: Example of synthesized Majority circuit with the consequent minimization and final result (using ME evaluation).

The synthesis process of Majority gate which was first realized by prof. Miller (as a permutative 3×3 reversible function) is presented in this section. Here our interest is only in single-qubit output Majority Gate. The Figure 5.10

shows a realization of the Majority gate found by the GA as well as the final circuit after minimization. The blocks denoted by numbers 1,2,3,4,5 can be each combined into a simpler gate. This is shown by transforming for instance the block 1 to the compound gate C_0 . The gates denoted by C_0 , C_1 , C_2 and C_3 are compound gates created by concatenating (collapsing) two or more gates that occupy the same wires and that do not have any other gate between other occupying other wires. For instance, the block 4 that is composed of $[NOTIC][CINOT][VIC]$ is transformed into a single quantum gate because all gates in it occupy the same qubits and no other gate in between them.

Observe that each step of the minimization, first minimizes the gates ($V \odot V = NOT$) and only then does the final gate concatenation. Thus at the end of the process, there are four blocks C_0, C_1, C_2, C_3 . Each block is created from $[NOTV^\dagger][NOTC]$ (on top two wires), $[CIV][CIV^\dagger][CIV^\dagger] \rightarrow [CIV][CINOT]$, $[NOTIC][CINOT][VIC]$ and $[CNOT][NOTC][VC]$ (on top two wires) respectively. Note that the block 5 ($[NOTIC][V^\dagger \otimes I]$) has been removed because the gates it contains are irrelevant to the single output function realized on the bottom wire.

5.3.3.1 Influence of t_{max} and t_{min} on the Evolutionary QLS

The Majority gate was successfully synthesized and designed using the GA with solutions close to the optimal one in cost. Similarly to the Fredkin gate synthesis, the Majority gate synthesis requires longer time (and is less success-

ful) than in the case of the Toffoli gate because the GA had more troubles to keep larger groups of gates together allowing to generate universal gates such as Toffoli or Fredkin which are required to build the Majority gate.

Figure 5.11 illustrates the Evolutionary Synthesis process of the Majority gate for various settings for the minimal and maximal sizes of the synthesized circuit, t_{min}, t_{max} respectively. Observe, that as expected (and as already observed in the cases of Toffoli and Fredkin), the relation of the prior knowledge about the desired circuit size directly affects the results: for larger circuits the precise settings of the same cost and t_{min}, t_{max} parameters are much more important than for the smaller circuits. Also observe, that the problem space represented by the bounds given by the minimal t_{min} and maximal t_{max} numbers of segments must contain circuits with the desired minimal cost.

From an experimental point of view, the size of the problem space given by the t_{min}, t_{max} parameters was observed to be inversely proportional to the number of solutions found. This can be seen for instance in Figure 5.7 which shows that for a given cost, the GA will be generating circuit solutions when the relation between the desired minimal cost, and the bounds t_{min}, t_{max} are both such that for a minimal cost c_m , there is at least one global solution (circuit) that has a cost $c \geq c_m$ and its size is $t_{min} \geq t \geq t_{max}$.

In the present experiment, the evolutionary search using $(t_{min}, t_{max}) = (5, 11)$ (Figure 5.11a) and $(t_{min}, t_{max}) = (13, 29)$ (Figure 5.11c) respectively under- and over-estimates the appropriate circuit size, while the search using $(t_{min}, t_{max}) =$

(a) Partial set S_p , $t_{min} = 5$, $t_{max} = 11$ (b) Partial set S_p , $t_{min} = 7$, $t_{max} = 15$ (c) Partial set S_p , $t_{min} = 13$, $t_{max} = 23$

Figure 5.11: Comparison of three different setting for minimal (t_{min}) and maximal (t_{max} number of circuit segments during the search for Majority gate using the single-qubit ME.

(7, 15) (Figure 5.11a) was successful.

The goal of manipulating the t_{min}, t_{max} parameters was to restrict the problem space to a minimal size in order to make the search computationally tractable. Another of the parameters that was used is the size of the circuit; i.e. the maximum number of parallel segments in an Individual. For instance, the GA starts with the minimal number of segments set up to 2 and maximal number of segments set up to 5. Note that in order to successfully design a gate the desired minimal cost must be below the optimum minimal cost.

The range of the size of the acceptable circuit size (defined by $l = t_{max} - t_{min}$) was also considered as a factor during the evolutionary process. This is because when l is small ($l < \epsilon$), then during the process of crossover, many circuits have to be repaired and information is lost too early in the evolutionary process (any repair mechanism naturally entails information loss). On the other hand, a large value of parameter l has an opposite effect and that is to search circuits with larger cost, despite the fact that the solution might be much less expensive.

The importance of the t_{min}, t_{max} parameters (as well as of l) on the synthesis process is clearly observable during the evolutionary search. When looking for a circuit of a known size n (and assuming this size is minimal) it is required to allow the GA to search spaces with chromosome length above and below the user specified circuit size. From previous work in this area [LPMP02, LPG⁺04] it was shown that a messy algorithm (with random size of the individual

Figure 5.12: Example Majority circuit found by the GA. The Figure also shows steps of the minimization process. Observe that the aggregate gate C_0 is created by moving the $CV^\dagger$ gate near the CNOT gate (left-to-right arrow in the middle circuit).

circuit (gene)) was not successful. In the here studied evolutionary synthesis method, the size of the circuit is much more controlled, however it is required that the initial estimate of the maximal size of the circuit appropriately overestimates the minimal size (or the expected minimal size) of the circuit. This is important because it allows to design given functions with various costs allowing different results close to minimum. Thus, the search space has to be restricted around the expected minimum as close as possible.

Similarly to the synthesis of Toffoli or Fredkin gates, some circuits obtained during the synthesis of Majority gate have the output qubit deterministic despite the fact that some other qubits can be still in a probabilistic state. One of such circuits is shown in Figure 5.12. Observe that again the circuit is min-

imized by concatenating neighbor gates defined on same qubits. For instance on the bottom qubit, the sequence of single qubit operations $[V^\dagger][NOT][V^\dagger]$ reduces to Identity and thus is removed.

Figure 5.13: Example of successful synthesis of the Majority gate using a partial input gate-set S_p and a single-qubit ME method.

Figure 5.13 shows a successful search for the Majority gate. Observe that similarly to the synthesis of Fredkin and Toffoli the evolutionary process finds the solution in one step. This means that the fitness function during the search reaches some local optimum and then in one step jumps to value one.

5.4 Discussion of the Results of the Evolutionary Quantum Logic Synthesis using *EE* evaluation

5.4.1 Toffoli Gate

Figure 5.14 represents typical circuits that have been obtained during the synthesis process using the S_f and the S_p input gate-sets.

Figure 5.14: Various realizations of Toffoli gate obtained from GA before their optimizing transformations were applied. (a) - three CNOT gates (the dashed frame 1) are collapsed into a single one, (b) - the two Z gates on the topmost wire can be removed as they do not change the observable of the circuit (after measurement), (c) - the frame 1 is collapsed to a single two-qubit gate, frame 2 is removed as $[CNOT] \times [CNOT] = [I]$ and frame 3 is collapsed to a single CNOT gate, (d) - is minimized to the well known minimal Toffoli gate by removing the two NOT gates and transforming two CV gates into a single CNOT.

The first circuit (5.14a) is the result of a run where fitness from equation 4.15 was used and a complete starting set of gates. As can be seen, this result is very expensive because it includes three Fredkin gates and four Feynman gates.

This circuit can be simplified by removing two of the three subsequent identical Feynman gates based on the fact that Feynman gate is its own inverse.

The next one, Figure 5.14b, is a result of the same setting as with the previous one but the starting set was a biased one. The available gates in this biased set were only the ones with number of inputs smaller or equal to 2 qubits. Similarly to the previous circuit, the group of gates in dash-squares can be concatenated in order to reduce the cost of the circuit. Moreover two pairs of consecutive identical Feynman gates (in dotted groups) can be removed. Thus, the second from left dotted group is removed entirely and the first from right dotted group is replaced by a single Feynman gate. The rightmost Feynman gate can be moved before the last Controlled-V gate and added to the dotted rectangle. Although this circuit is longer than the first one its cost is the same. It is composed only from 2-qubit or 1-qubit gates. The first circuit has cost 67 ($5*2 + 19*3$) after minimization and the second circuit after minimization has cost of 45 ($9*5$).

The two circuits on the bottom of Figure 5.14 were found using the improved fitness function from equation 4.15 and 4.17. The circuits from Figures 5.14a, b and c were found using the complete starting set S_f . The circuit from Figure 5.14d was found using the biased set S_p . The major improvement using the equation 4.17 as the fitness function is that in a well delimited problem space this fitness function can speed up the search and possibly find the optimal gate. However as is discussed in Section 5.5 this improvement is limited to the search for less expensive gates as in general it results in longer runs of

the GA. Observe that by flipping over two Feynman gates from Figure 5.14c and removing the swap gates a solution with one Fredkin and two Feynman gates is generated, which is close to the known realization of Toffoli gate from Fredkin gate. The only difference are two Pauli-Z rotation gates, which can be removed, as analysis shows.

The results from Figure 5.14 have been obtained using the EE method. However, some of the exact benchmarks have also been synthesized using the ME evaluation method. In the case of the Toffoli gate, the gate can be specified by observed values only on desired qubits; a single output function.

Interestingly, using the EE method (as well as using the ME) method, our software reinvented also the famous circuit of Smolin [SD96], since the sequence of two Pauli-X (NOT gates) can be removed, and the sequence of two Controlled-V is equivalent to Feynman gate (Figure 5.14d). After these transformations, our gate is composed from the same basic quantum primitives as the famous (and used in most designs and papers to realize quantum algorithms) Smolin's solution [SD96], but in a different order. The comparison of the unitary matrices shows however that the circuit invented by my GA and the circuit invented by Smolin are equivalent. These examples show that not only can our software "reinvent" the realization of the known gates but it can also create similar minimum cost realizations of other gates, as will be presented in chapter 6. We were not able to find a better solution to Toffoli and Fredkin gates from quantum primitives, perhaps they do not exist.

5.4.2 Fredkin Gate

Figure 5.15 presents some of many found realizations of the Fredkin gate using EE evaluation, the fitness function from eq. 4.15, the full set of gates S_f . Figure 5.15a and Figure 5.15b show the result when the full input gate-set S_f was used. Figure 5.15c and Figure 5.15d show results when the input gate-set defined by $S_f - [SWAP]$ was used.

Figure 5.15: Results of synthesis of Fredkin gate. a) is result of application of fitness from eq. 4.15 and a complete set and b) the result of fitness from 4.17 and a complete set. Figure c) and d) represents results obtained using the same settings but without using the SWAP gate in the input gate-set.

Again, observe that most of the circuits can be minimized to the well known optimal Fredkin solution. The consecutive Pauli-Z gates can be canceled (Figure 5.15b) since this gate is its own inverse (known as a standard local transformation). Next, in both Figure 5.15a and Figure 5.15b the swap gates can be removed by manipulating the circuit: changing the ab controlled Toffoli to ac

controlled Toffoli (Figure 5.15a) changing to two Feynman gates, by flipping them upside down (CNOT gates to NOTC (bottom-up CNOT)) removing the SWAP gates. This leads to the known (minimal) realization of Fredkin gate using one Toffoli and two Feynman gates (Figure 5.15b). Also observe that the gates from Figures 5.15c and Figure 5.15d are easily transformed into the minimal solution, by simply removing the single qubit gates $[Z]$ and $[S]$. This does not have any effect on the resulting function as the corresponding gates manipulate only the phase of the quantum state.

One of the observations of the obtained results is the respective presence of Fredkin and Toffoli gates in the circuits representing Toffoli and Fredkin respectively. In fact, using one of the two gates does experimentally guarantee finding a result, while not providing such a universal gate highly reduces the chances of finding the correct gate. As can be seen in the results of synthesis using the complete starting set, the Fredkin gate is present in all Toffoli implementations and vice versa. Without a universal gate in the input gate-set the GA has troubles to form and preserve during the evolutionary process such segments on multiple qubits. It can be concluded that providing more complex universal gates allows the GA to overcome a local functional minimum.

5.4.3 Entanglement Circuit

Another set of benchmarks was the entanglement circuit for two, three and four qubits. Figure 5.16 shows the results of synthesis of the two-qubit en-

tanglement circuit. Observe that both presented examples can be minimized to the well known entanglement circuits. Also, as described in chapter 4, two different implementations of the entanglement circuit can be observed for two and three qubits: using the well known Hadamard gate and using the V gate (Figure 5.16 and 5.17).

Figure 5.16: Results of Synthesis and Minimization of the Entanglement circuit for 2 qubits.

Figure 5.17d shows an entanglement circuit for three qubit realized only CNOT and V gate (the NOT gates can be removed without changing the overall circuit output). It is interesting to observe that this circuit (as well as circuit in Figure 5.16a) generates complex entangled states. These states will be indistinguishable under measurement from the Hadamard based circuit generated entangled states.

Observe, that most of the gates synthesized can be minimized to the well known entanglement circuit not because the presented results are directly a sub-circuits of such gates but rather because most of component gates in the presented circuits can be removed. Such action will modify the possible entangled output states but will preserve the entanglement.

Figure 5.17: Results of Synthesis and Minimization of the Entanglement circuit for 3 qubits.

Figure 5.18 illustrates the four-qubit entanglement circuits together with the minimization process. Similarly to the case of two and three qubits, half of the circuit can be thrown away in the case when the goal is to obtain any entanglement circuit. Observe that under such assumption (when the only goal is to obtain any type of entanglement), all of the circuits (Figure 5.18) have the same cost when minimized.

Figure 5.18: Results of Synthesis and Minimization of the Entanglement circuit for 4 qubits.

5.5 Discussion of the results of the Evolutionary Synthesis

5.5.1 Results Comparison

Table 5.5.1 presents a summary of the results in QC search for the benchmarks used in this dissertation. The first column is the name of the function searched, the second column shows if the software solution was found using either the non-optimized fitness function (eq. 4.13), or the optimized fitness function (eq. 4.17). The third column shows, for each circuit, the solution times in terms of numbers of generations necessary to arrive at a solution. Each run was stopped after it reached the maximal generation count in the cases for which no solution was found previously in the run.

As can be seen in five cases out of the seven test benchmark problems (Toffoli, Fredkin, Miller and Entangle4), a significant time increase was observed when using the fitness function from eq. 4.17 (Table 5.5.1 rows 1,2,4 and 7). Two

Table 5.4: Comparison of various fitness functions for all illustrated benchmark functions

<i>Problem Name</i>	<i>Solution</i> [Fitness (eq. 4.13)/(eq. 4.17)]	<i># Generations</i> [Fitness (eq. 4.13)/(eq. 4.17)]
Toffoli	YES/YES	< 200/ < 500
Fredkin	YES/YES	< 200/ < 750
Majority	YES/NO	< 100/ < 300
Miller	YES/NO	< 400/ < 800
Entangle2	YES/YES	< 100/ < 100
Entangle3	YES/YES	< 100/ < 150
Entangle4	YES/YES	< 200/ < 400

arguments support the use of the improved fitness function (eq. 4.17). First, the solution was found in all cases and it was found even when using a biased set. This argument implies that the cost function allows the GA to preserve individuals in the population that would be otherwise discarded and this allow to preserve the diversity in the population. Second, the obtained results using the fitness function (eq. 4.17) in general generated shorter circuits as well as circuits either optimal or close to the known optimal circuits.

5.5.2 Encountered problems during the Evolutionary QLS

As is described in this chapter, the GA was able to find quickly circuits for two, three and also for four qubits, but was much slower for larger gates and circuits. The reason for this lower performance is due to following problems:

- the size of input gate-set
- the allowed size of the circuit to be searched

- the level of the element gates
- the relation between the error evaluation and the cost of the circuit

The size of the input-gate set was already discussed and analyzed in section 5.3.2 as well as the size of the problem space was analyzed in section 5.3.3. The last two points to be discussed are thus only the impact of the level of the synthesis on the performance of the GA (section 5.5.2.1) and the relation between the error evaluation and the cost function (section 5.5.2.2).

5.5.2.1 The level of Synthesis and the Pulse-based synthesis

Figure 5.19: Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.

In the case of the pulse-based synthesis (input set S_{lr}) the search was more difficult both in the case of the Toffoli gate but also for all other benchmarks. In fact as will be seen, the synthesis using the unitary pulses of arbitrary (user limited) angles was mostly unsuccessful.

Figures 5.19 and 5.19 represent some of the results using the GA. The rotation was parameterized by the set $\{\pi, -\pi, \frac{\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{4}, -\frac{\pi}{4}\}$. One of the particular

observations on the pulse-based synthesis is that the algorithm found during these experiments mainly only approximate gates in the allotted time.

$$R_0 = \frac{\pi}{2}_z \cdot \frac{-\pi}{4}_y \cdot -2\frac{\pi}{4}_x$$

$$R_1 = \frac{\pi}{2}_z \cdot \frac{\pi}{4}_y$$

Figure 5.20: Example of sub-optimal approximate Toffoli using the S_{lr} input gate set.

The lack of success when synthesizing using the angle-parameterized unitary rotations is because

- the GA has troubles to keep many small unitary rotations together in order to preserve higher level gates such as Toffoli or Fredkin
- the input-gate set grows too fast in size and thus to obtain a desired circuit the GA requires both more time and computation

5.5.2.2 The Cost-Error ratio

Figure 5.21 shows the recording of one run searching for the Fredkin gate using the fitness function 4.15. For illustration the fitness 4.17 is also drawn. The figure shows the best result of each hundred generations. The first curve Error shows the evolution of the scaled error as used in the fitness function. Second

Figure 5.21: A comparative chart between two different fitness functions during the same GA run.

curve is the cost ($1/\text{Cost}$) and it is to be maximized to increase the global fitness function. The larger a circuit the more the cost is reduced. The last three curves show the values for fitness from equations 4.15 and 4.17. Fitness function from eq. 4.17 is shown by two curves; first with parameters $\alpha = 0.99$ and $\beta = 0.01$ then with $\alpha = 0.9$ and $\beta = 0.1$. For more clarity all fitness functions are mapped on the secondary y-axis on the right.

The Fitness function from eq. 4.17 with ($\alpha = 0.9$) captures mainly the correctness of the circuit as well as the cost ($\beta = 0.1$). Such configuration however requires a large final jump to the correct solution (≈ 0.1). Such a fitness function can be used to minimize the size constraints but might not be the best candidate to find circuits with minimal error.

The best fitness function to capture global properties during a search for a correct circuit is the fitness function from eq. 4.17 with ($\alpha = 0.99$). This is because on one hand $\alpha = 0.99$ captures the correctness of the circuit and on

the other hand a $\beta = 0.01$ takes into account the size of the circuit in such a way that allows both to search for the correct circuit and minimize the size (cost) of it in an appropriate ratio. Consequently, comparing it to the fitness function (eq. 4.15) one can observe similar result but with small oscillations in the fitness function (eq. 4.17 with $(\alpha = 0.99)$).

The Figure 5.21 illustrates also the general slow convergence of the GA. Observe that this can be seen on Figures 5.3, 5.9 and 5.13 where the results appears as a jump in the fitness value rather than as a result of a convergence of the fitness value to the value of one. The reason for this is, is the structure of the error that is always symmetric. In other words because the evolved circuit is a unitary matrix built from unitary matrices (quantum gates) any discrepancy between the target and the synthesized circuit will be always on 2^r coefficients.

This also means that when synthesizing using only permutative component gates, the magnitude of the error will always be at least $\frac{2}{2^n}$ which results in the fact that the GA overall fitness will be stuck very often on a local maximum before in one step reaching the fitness value of one and error value of 0. In the case when arbitrary quantum gates are used for synthesis the magnitude of the error can be arbitrarily small. However because in the present case we are using only a set of selected quantum gates the error that can be generated is again bounded by the used gates and thus confirming that the GA reaches the correct solution (fitness = 1) after a jump from a local optimal fitness value.

5.6 Conclusion of the Evolutionary QLS

This chapter presented an innovative approach to quantum logic synthesis using a fitness/error function and a circuit cost. Similarly to previous work in this area, our algorithm was able to successfully use the selected cost policy and synthesis constraints to generate new circuits implementing permutative reversible circuits.

An interesting result of the evolutionary search is that while using the NMR pulses for the logic synthesis was not very successful for three-qubit circuits, the logic synthesis using the joint approach through pseudo-binary and unitary pulses allowed to find new circuits, otherwise not possible to discover on two qubits. This was not achieved by any of the previous authors.

The Figure 3.11 and 3.12 shows the implementation of the CNOT and CV gates using NMR pulses, that have been designed by our GA algorithm. Because the I_{zz} does not commute, the circuit is split in half. Thus, two sides of the quantum circuit can be distinguished; but this observation is only possible when looking into the sub-permutative-logic level. Similar observation was made with the synthesis model using only the $CV/CV^\dagger$ and $CNOT$ gates. The usefulness of this anti-symmetric property is the fact that circuits with a given structure can have different function when the order of the component gates is inverted.

As conclusion the following observations can be made:

Figure 5.22: Inversely constructed CNOT generates CE

- The cost-error ratio is a difficult problem that can be partially solved using evolutionary QLS
- The QLS as evolutionary process is sensitive to multiple factors such as the size of the input-gate set, the width of the synthesized circuit, the level of synthesis, the ratio between user specified minimal cost and the desired circuit size. Together, these problems makes the evolutionary QLS a hard problem
- The evolutionary QLS allowed us to rediscover already known realizations of quantum circuits (Fredkin, Toffoli) as well as invent novel interesting gates such as Entanglement, Majority, etc.
- The ME and EE methods of evaluations are quite useful insights into the QLS from an algorithmic point of view and allowed us to discover novel, partial realization of universal quantum gates.

6 STRUCTURE BASED SEARCH FOR UNIVERSAL QUANTUM CIRCUITS

6.1 *Introduction*

The observations from the Evolutionary search showed that the best results for search are given by knowing the exact minimal cost of the desired circuit and by an appropriately sized search space. The observation of the fact that Evolutionary Synthesis got often stuck in a local minimum and a deeper analysis of this phenomena allowed us to make a structural discovery. The results of the GA (when the desired gate was not found) were often quite similar in the structure of the circuit. In particular, often the circuits would have components used in Peres or Toffoli gate but in a wrong order. Because the cost of these 'side-product' gates invented by our algorithm was also often quite similar and even close to the minimal cost (considering the cheapest universal quantum gate Peres), the natural continuation of this search algorithm was to explore the structure of a given circuit with a very small number of quantum gates.

Thus, one could look at the problem of QL synthesis as: given a cost, build a desired function. In such case, not only the number of gates is limited, but the overall structure of the circuit is preserved. For instance, each of the universal gates studied here (especially the least expensive one - Peres) is represented by

a particular set of gates in a particular order and is built using only two-qubit gates. Also, one can simply look at the circuit as an entity built from non-changeable segments, and only permuting them will generate circuits with the exact same component gates but with different functions realized.

The initial use of this search method (called EX), was used as an extension to the previously described GA. The combination of the GA with the EX algorithm is called the GAEX. However, GAEX was not successful in creating novel circuits and helping the synthesis process of the GA, because a) for large circuits the EX algorithm is not tractable and b) even for small circuits this search would only rarely create such circuit that would lead to a solution. On the other hand, the GAEX search allowed to demonstrate that simply permuting segments of gates, one can create different functions described by a set of well specified quantum gates.

In the GA input file the parameter *Local Search generational switch* (line 03, pseudocode 4.1) indicates the minimum number of GA generations to be executed before starting the EX algorithm. Thus as long as the termination condition was not reached the combined algorithm GAEX stops every k generations and initializes the EX algorithm for a limited period of time the EX algorithm. Once the allocated time is over or when the EX algorithm terminates, the results are inserted in the population and the evolutionary process starts again.

A schematic representation of GAEX operation is shown in Figure 6.1. On the

Figure 6.1: Figure (a) illustrates an unsuccessful GA run with the final solution having fitness value $\sim 80\%$ and a cost close to the minimal known cost. In such a case one of the solutions generated during this run can be used as a template for the exhaustive search. This is represented in (b) where each line is the path of a GA run through the problem landscape, with a deeper exhaustive search at the end (represented by a circle). The axes in (b) are representing a parametric space defined by the possible quantum circuits that the GA can synthesize during the evolutionary search.

left side (Figure 6.1a), an example of a unsuccessful run. In the Figure 6.1b, five different GA runs are represented by lines indexed by r_0, r_1, r_2, r_3, r_4 on the parameter space. The two axes represent the space of all possible circuits on which each GA run is represented by a line representing the starting point, the path through the problem space and the final point. The final point is located within a circle, representing the fact that at that point in the experiment the EX algorithm performs a local search. Each time a run is not successful the EX algorithm can be run in order to search for possible solutions.

6.2 The EX algorithm

The EX is a very simple and very fast search algorithm. It does not provide any heuristics¹ and therefore it is used only within a larger algorithm and only on a small problem space. It can be treated as "knowledge based" heuristic operator within the GA.

Figure 6.2: Example of segment encoding for exhaustive search. Left bottom is an example of encoding segments to gates while right bottom shows an example where each gate becomes a segment.

Each time the EX algorithm is used, it accepts as input a quantum circuit built from a set of gates. The pseudo code of the EX Algorithm is shown in pseudocode 6.1. EX offers three main modes of search that are the pseudo-random (shake), permute gates from circuit (permute) and rotate gates in a circuit (rotate). When used with the GA the algorithm performs a search and the best resulting circuit is reinserted into the GA population in the cases

¹ Here exhaustive means that the exploration of the complete space is required. The finding of the best solution is only a part of the entire problem solution

when the target circuit was not found. The EX algorithm accepts the best current individual and uses it as the initial state. The algorithm stops when the *termination-condition* is reached. The condition is either that the algorithm reached the maximum number of operations or a better (than the initial one) circuit was found.

For short circuits an exhaustive counter based permutation of circuits is done until the initial circuit is found. For large circuits, the algorithm starts as in the Short Circuit case and after $k \cdot n$ for $k = 1, 2, \dots, r$ attempts a mode at random from the three possible modes. Each call to a function on the circuit is time limited and once the limit is reached the algorithm returns to a new set of iterations of exhaustive counter-based permutation.

Table 6.1: Pseudo Code of EX Algorithm

```

01:   $t \leftarrow 0$ ;
02:  initialize(Qc);                                /* start the initial circuit
                                                    and evaluate */
03:  for (k) for m permute(Qc);                      /* perform exhaustive counter-
                                                    search for k steps*/
04:  while (!termination-condition) do
06:    switch(s = random())
07:      case: 0  $\rightarrow$  shake(Qc);                  /* randomize by a pseudo-random
                                                    permutations*/
08:      case: 1  $\rightarrow$  permute(Qc);                /* randomize by a permutations
                                                    for m steps */
09:      case: 2  $\rightarrow$  rotate(Qc);                /* randomize the circuit by
                                                    swapping gates in symmetry*/
10:  end while

```

The *shake(Qc)* operation randomly swaps all gates on the circuits with each other. The *permute(Qc)* operation takes all the segments of the circuits and

performs all possible permutations of this circuit - if allowed by enough time. The *rotate(Qc)* operation swapped gates that are symmetrically opposite in the circuit.

The algorithm operates on circuits that are represented as parallel blocks. Thus depending on the specifications, each block can either be a single gate or multiple gates in parallel; in both cases a gate swap means a swap of segments of equal width (full width of the circuit): $|\psi\rangle = U_i \otimes \cdots \otimes U_l$.

An example of these segmentations can be seen on Figure 6.2 where the original circuit built from five gates is re-encoded to either four maximum-width or five maximum-width unitary transformations. Such a setup permutes the gates in the circuit. It also preserves the overall structure of the original circuit.

Because a complete exhaustive search for larger circuit benchmarks is not tractable, the EX algorithm allows to locally change the way how to search the problem space. In particular EX can examine all the permutations of gates from a limited set. It also allows to minimize and transform circuits with many gates by local changes and to synthesize all circuits with up to k gates.

Example 6.2.0.1 EX algorithm

The EX algorithm is capable of limited exhaustive search called 'structural constraints'. For this assume that a circuit is specified by the String $S = \{pCVIp\ pNOTCXp\ pVIIp\ pCV^\dagger Xp\}$ ² (Figure 6.3). The circuit is divided into segments such as $pVWWp$ or pC_VWp .

² The encoding is explained in details in Chapter 4

Figure 6.3: Random permutative circuit

The EX specified by the structure constraint will execute non repetitive permutation on every single gate of the circuit on all wires but using either only the initial configuration of the gates or allowing some user-specified changes. The selected example was found when the EX algorithm specification was gate-type dependent; only a certain type of gates can be altered by changing the wiring in the circuit. This means that given the circuit from Figure 6.3a only the CNOT gate was selected to be exhaustively altered and placed in all possible positions in the circuit resulting in Figure 6.3b.

It is interesting to notice that beside the fact that only one gate changes position and configuration in the circuit, the prototype on the left side (a) has a non permutative matrix while the circuit at right (b) is a permutative circuit which can be further reduced by collapsing the CV and $CV^\dagger$ on the middle wire[LPG⁺03]. This example illustrates how the EX algorithm can be used when searching for permutative circuits.

Note, that unlike the GA which was exploring the *structure space* of circuit with the secondary goal the cost minimization, the EX algorithm explores the *cost space* of the circuit with secondary goal to discover permutative quantum-logic circuit. $\square$

6.3 Heuristic Search for Lowest Cost Function Class using the EX algorithm

The exploration was realized using the exhaustive search of all possible permutations of component gates (circuit segments) of the least expensive of quantum universal 3-qubit gate, the Peres gate. The choice for Peres gate has the following reasons. Peres gate was shown extremely useful in synthesis [YSHP05], and it was also proven to be a part of the minimal sets and the least expensive sets of universal quantum gates [BBC⁺95]. Also, the Peres gate is used in the implementations of Toffoli, Fredkin, Majority and many other known gates by simply being surrounded by some affine pre- or post-processor circuits.

The exploration of these observations in more depth is a logical extension of the previously presented evolutionary search. Also, the here presented approach will hopefully reveal information about the structure of small quantum circuits. This approach will be useful for quantum logic synthesis [MD04, HSY⁺04, YSHP05]. The exhaustive search introduced in this section is based on the generalization of the structure of Peres gate. The search is intended to discover new cheapest quantum universal gates by cost minimization and by applying alterations on the circuit that preserve most of its useful features (universality, functionality, synthesis properties). Similar approaches can be formulated for other gates than Peres as well.

The logic function of Peres gate is given in Table 6.3. The well-known Peres gate realization has four two qubit gates: two controlled V, one controlled $V^\dagger$ and one Feynman gate. The function of this gate is $\{a, b, c\} \rightarrow \{a, a \oplus b, ab \oplus c\}$.

Table 6.2: The Peres gate Truth Table

abc	$a^+b^+c^+$
000	000
001	001
010	010
011	011
100	110
101	111
110	101
111	100

Figure 6.4: All possibilities of placing a Feynman gate on three qubits

The EX algorithm was used to search for alternatives based on the template of Peres gate, using only the primitives from the Peres itself in every possible combination and permutation. The generated set of gates started with a circuit composed from a single gate and ended with four gates in all possible combinations of CV , $CV^\dagger$ and Feynman gate. In other words, the EX algorithm searched all possible combinations of up to four gates in series [LPG⁺04]. Each used gate can be placed in six different ways in a 3 wire circuit, as shown for the Feynman gate example in Figure 6.4.

The total space spawned by the EX algorithm can be described by approximately $(18)^4$ alternative circuits. Most of the generated circuits have not been

recorded because the realized circuits generate probabilities different than only 0 and 1 in values (these circuits were in 4-valued quantum logic being a superset of binary quantum logic [HSY⁺04] and a subset of the general quantum logic).

6.3.1 New Quantum Logic Family

The most interesting results of the EX algorithm based synthesis are recorded and their logic equations are shown in Table 6.3. These reported results are minimized and only non-trivial function circuits are presented. Also for each obtained result the circuit is drawn and a schematic diagram is provided.

The cost of any of these functions is the same as that of the Peres gate, which is 5. In particular, all shown gates have 4 two-qubit gate primitives. The table reports four functions. However, the set of generated permutative circuits produces the permutations of the output functions on the wires as well as the permutation of literals. Table 6.3 shows recorded non-trivial and interesting results for future synthesis algorithms because of their low cost. Thus functions as constant 0, constant 1, variable or negated variable are not shown because they can be realized with a smaller cost than the Peres gate. Interestingly,

Figure 6.5: (a) Circuit representing function $f(a, b, c) = [0, 1, 2, 3, 7, 6, 4, 5]$, (b) Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 6, 7, 4, 5]$

Figure 6.6: (a) Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 7, 6, 5, 4]$, (b) Circuit representing function $f(a, b, c) = [0, 1, 3, 2, 7, 6, 4, 5]$

more circuits of the form $a \oplus (b + c)$ were not generated. It is however easily seen that such functions can be obtained using one more inverter or two V gates in series (V gate is the square-root-of-not gate).

Table 6.3 shows functions of the form $A, B \oplus C, A \oplus BC$, where A, B and C are literals of arbitrary polarity. The realization of the presented functions can be observed on Figures 6.5 to 6.6, and their logic diagram in Figures 6.8 to 6.9.

The realization of the above described Peres-based gates makes the presented gates the cheapest family of universal permutative gates. The generalization of this family of gates is also known as the Perkowski gate. Some gates of the Perkowski type gate family [LPG⁺04, LPMP02] are schematically represented

abc	<i>Permutations</i>	<i>function</i>
Peres	0 1 2 3 7 6 4 5	$a, a \oplus b, c \oplus ab$
	0 1 2 3 7 6 4 5	$a, a \oplus b, c \oplus ab'$
	0 1 3 2 6 7 4 5	$a, a \oplus b, c \oplus a'b$
	0 1 3 2 7 6 5 4	$a, a \oplus b, c \oplus (a + b)$
	0 1 3 2 7 6 4 5	$a, a \oplus b, a \oplus b \oplus c$

Table 6.3: The family of Peres gates

in Figure 6.8 to Figure 6.9. Thus the results of this dissertation strengthens the arguments for using Peres-like gates as a basis for synthesis because of their cost and general properties. As the Peres gate family is only a subset of the Perkowski type gate family every gate of Perkowski gate family can be used similarly as Peres gate to synthesize inexpensive circuits [LPMP02]. This possibility has been not yet investigated and remains an open problem for GA and GAEX.

As a result of this research an important question appears. *What is the relation between the cost of the logic operators AND and OR as related to the realization in quantum technology using only $CV/CV^\dagger$ and CNOT quantum primitives?* As already mentioned the CNOT gate is derived from the CV. Thus this approach can be reduced to permuting only CV and $CV^\dagger$ gates.

One of the interesting observations from this controlled search is that using particular gate type in a fixed structure and changing just the controlled unitary transform (changing Controlled-V to Controlled- $V^\dagger$), allows to create Peres gates with different polarities of inputs. Figures 6.5 and 6.6a show that by only manipulating the ratio and the placement of CV and $CV^\dagger$ the various Peres-like functions have been obtained. This is quite remarkable, because we are creating new classical functions by only manipulating the order of the purely quantum gates. Also the synthesis method can be described by synthesizing a control function and the controlled unitary transformation placement in the circuit at the same time. The placement is the structure of the circuit from Figure 6.5; $[W, CU][CNOT, W][W, CU][CU]$ - four three qubit wide quan-

Figure 6.7: Circuit realizing the reversible function $f = [0, 1, 3, 2, 7, 6, 4, 5]$. Each CNOT gate was broken down to two CV gates.

tum gates where CU is replaced by an desired operator. Also observe that similar structure is obtained when the circuit from Figure 6.6b is broken to Controlled-V gates (Figure 6.7). This observation is explored in next Chapter, where it is the basis of the Quantum Symbolic Logic Synthesis.

Finally the efficiency of a simple approach such as a locally applied exhaustive search must be appreciated as useful and powerful. The idea here was not only to simply search a randomly created space of quantum gates. Our idea is to search a "useful" space so as even with smaller resources the results are powerful and useful; both for the practical synthesis and also for the theoretical insight. Usually this space is determined by some gate solutions found by the GA. It must be mentioned that the complete run of the EX software is not time consuming. The EX algorithm gives results with quality proportional to

Figure 6.8: Circuit representing function $[0, 1, 3, 2, 6, 7, 4, 5]$, Circuit representing function $[0, 1, 3, 2, 7, 6, 5, 4]$

Figure 6.9: Circuit representing function $[0, 1, 2, 3, 7, 6, 4, 5]$, Circuit representing function $[0, 1, 2, 3, 6, 7, 5, 4]$

the smartness of the given problem formulation. The Peres gate is an ideal candidate for this approach however other candidates can be used such as Miller gate or Fredkin gate.

6.4 Discussion and Conclusion to the Structure Based Exhaustive search

We showed that the EX and the GAEX algorithms that combine the logic correctness, the physical implementation cost and some search heuristics are able to find families of new interesting gates. Our new approach discovered already few previously unknown permutative gates that are useful in systematic synthesis algorithms for reversible logic [KPK02, MD04, MM04, PARK⁺01, SBM05a, YSPH05, YSHP05, MP02]. The goal of this research was the discovery of optimal and quasi-optimal generalizations of the Peres, Toffoli, Fredkin, Miller gate [LPG⁺03] for $n > 3$ (to be used as elements of larger gates) to be used in wave cascades and similar cascades design as in [PARK⁺01, KPK02, MP02].

The results of our approach proved the following points:

- the generation of reversible functions from a given universal primitive circuit allowed to generate a whole family of gates with the same cost and similar functionality.
- the Peres gate family is not only the Peres gate generalization but also a very low cost full featured universal set quantum gates
- reducing the quantum problem space to a small subset (three quantum gates) and limiting the size of the circuit allowed us to generate an exhaustive algorithm that would not be possible for larger quantum circuits/gates
- the Peres gate family can be used to design cascades such as Maitra cascades as it provides both the logic OR and the logic AND operations.

Another interesting result was the automated discovery of the general property of the CNOT-CNOT block. Such a block is based on two CNOT gates. In between these two gates another gate can be inserted; for example the NOTC³ gate generates the SWAP gate. Similarly, the GAEX algorithm constructed the T gate by inserting a VC gate between the gates of block CNOT-CNOT giving $[CNOT][VC][CNOT]$.

These results give interesting properties of quantum circuits realized using gates of type V, $V^\dagger$, T and $T^\dagger$. That is, so far the methodology to create

³ NOTC gate means that the NOT bit is on the first line, the control on the second line.

Figure 6.10: Permutative circuits obtained using only CV and CNOT gates. a) Observe that only gate CV is used, thus a new principle of creating permutative gate was discovered by the EX algorithm, b) by simply adding a Feynman gate, the circuit realizes now both a multiplication and addition.

permutative circuits popularly accepted in QLS community [SD96, BBC⁺95] was to use a combination of a gate and its conjugate such as V and $V^\dagger$. However, the circuit in Figure 6.10 offers an alternative: a judicious placement of CNOT gate allows to eliminate $V^\dagger$ gates from the data path. This operation results in a permutative circuit. This elimination makes it possible to perform more exploration of various technologies and various constraints in order to find which type of synthesis is cheaper and more efficient (V and $V^\dagger$ cost the same in NMR, but the discovered here principle is more general).

Figure 6.11: Trivial alteration of the circuit from Figure 6.10 (changing a single CV gate to $CV^\dagger$ and adding an inverter on qubit a).

These first preliminary results allowed us to generalize the principle from the Peres family gates; Figure 6.11 and Figure 6.12 demonstrate how moving around the $V/V^\dagger$ gates allow to generate permutative and non permutative quantum circuits and keeping the same circuit structure. This is quite unusual, as in general different functions require different circuits, not only dif-

Figure 6.12: Inverse non trivial alteration of the function $[0, 1, 3, 2, 7, 6, 5, 4]$ (generating a new function on the circuit outputs)

ferent controller single qubit unitary transformations. Observe that circuit from Figure 6.12 uses the same segments as the circuit from Figure 6.11 but in the inverse order and with the location of the $V^\dagger$ transformation moved. The consequence is that the original functions on the lower two qubits $a \oplus b$ and $ab \oplus c$ is changed into $a \odot b$ and $a \odot c$ (with $\odot$ being the NCNOT function).

The results from the EX experiments are encouraging because this is the first time that a program found a gate family from quantum primitives where this family is more general and has the same cost as a single gate found earlier by a human (prof. Peres). Moreover the exhaustive algorithm is a proof of minimality of the Logic Family. Finally the bottom line is the observation and the conclusion that the construction of quantum circuits should no more be done using Fredkin and Toffoli family of gates but rather the Peres gate family for 3-qubit circuits. However, the synthesis method using the Peres gate or even the family of the Peres gates, will be more difficult than using Toffoli gate. This added search complexity is due to the fact that the process of synthesizing with a two outputs versus a one output universal quantum gate requires more computation and search.

Moreover, the synthesis with a single-output functions has the following prop-

erties

1. recycles all control qubits and
2. because it has a single output, local algorithms and searches can be formulated more efficiently than in the case where two outputs of the gate have to be manipulated in order to restore all control qubits to the initial state and preserve the output qubit.

7 LEARNING QUANTUM BEHAVIORS

The automated evolutionary approach described in Chapter 5 showed the results and experimentation for the synthesis of completely determined universal quantum gates. As introduced in Chapter 3 a function can also be incompletely defined. Such incompletely defined problems are useful in the area of Machine Learning where Boolean functions with large number of don't cares are used to represent problems. For instance, taking example of a robot that learns how to interact with humans, the quality of its behavior depends on how well it can learn to generalize from experience (set of examples) to create appropriate actions in new and previously unknown situations.

Unlike Boolean functions, the robots express the control by a physical behavior. For the well defined tasks that occur in industrial and mobile robotics, the robot is controlled by a robust controller keeping its behavior well within specified parameters. In behavioral robotics, the goal is the interaction between a robot and human; robots are used to interact with children for learning, to learn the principles of social behaviors with humans in order to improve the task/data communication or to study human behavior in order to build a viable model of emotional or reactive behavior. The interesting point is that despite an overall goal orientation, a social robotic behavior cannot be described by an algorithm because the behavior would become *boring, inad-*

equate, inflexible and so on. Thus the specification of a robotic activity that would satisfy such criteria is an interesting model for learning behaviors using quantum logic synthesis.

In this chapter, the problem is to synthesize quantum robotic behaviors. The synthesized behaviors are represented by logic circuits and this formalization allows to apply logic synthesis and study the behavior as a black box. We use the GA to synthesize the quantum behaviors (circuits) for incompletely specified problems and we synthesize from examples where examples are cares of the quantum truth table. We analyze quantum inductive bias by identifying new states allowing the synthesis method to discover novel behavioral generalizations, for incompletely specified problems otherwise impossible to find in classical robotic behaviors. Finally we propose a symbolic method of analysis and we construct quantum circuits based on a measurement-based synthesis method. We also reanalyze some of the results obtained during the evolutionary synthesis for completely specified functions.

This chapter introduces Quantum robotic Behaviors in Section 7.1 and the process of Learning from Examples in Section 7.2. Section 7 presents the Symbolic Quantum Logic Synthesis methodology and explains the quantum symbolic synthesis as Machine learning of robotic behaviors from examples used for two types of synthesis: the quantum probabilistic synthesis and the measurement dependent synthesis. Each of the presented methods is illustrated by an example. Finally, a discussion of the presented concepts, application of this approach and conclusion are presented in Section 7.4.

7.1 Quantum Behaviors

As introduced, a robotic behavior is different from a function. The external inputs to a robotic behavior are sensors, and the outputs are physical actions. The behavior generating the outputs, is based on an internal state. A behavior thus is a set of robotic actions (outputs) based on a set of possible perceptions (inputs). This can be compared to the behavior of a simple animal or insect. It was Braitenberg [Bra86] that introduced this concept and described it by a set of simple input-output connections allowing to create various behaviors. For instance, robots with two sensors and two actuators can be easily connected to express either the behavior "*afraid of light*" or the behavior "*attracted to light*".

In Chapter 3 we introduced the fact that because a quantum system by default keeps its state until modified, it can be by a general extension seen as natural realization of a Finite State Machines (For more details on computational model in quantum computing see Appendix B). In the context of Behaviors, each Behavior represents a particular configuration of the FSM using the desired number of states, the state transition function and the set of states.

To represent a behavior, first assume that the robot's inputs are represented by i_1i_0 where 0 and 1 represent respectively off and on. The motor control is represented by outputs o_1o_0 with control states 0 for the reverse drive and 1 for the forward drive (it is assumed the wheels are always moving). This

allows to represent the two behaviors *afraid of light* or *attracted to light*, as shown in Table 7.1 and the appropriate circuits are shown in Figure 7.1.

Table 7.1: Braitenberg vehicle behaviors: (a) *afraid of light* , (b) *attracted to light* .

	$i_1 i_0$	$o_1 o_0$		$i_1 i_0$	$o_1 o_0$
	00	00		00	11
(a)	01	10	(b)	01	01
	10	01		10	10
	11	11		11	00

Such defined behaviors are however very simple and the behavior they generate is a function or a mapping, not allowing for any variation. A natural extension is to design probabilistic behaviors. In such a case each input state $i_1 i_0$ will generate an output $o_1 o_0$ with probability p_i . This probabilistic output generation is more powerful, however there is not a natural hardware representation for such a behavior as natural as is the Boolean logic for deterministic behaviors.

Figure 7.1: Circuits representing behaviors (a) *afraid of light* and *attracted to light* as shown in Table 7.1.

Extending the behaviors to quantum is just another step toward a *natural* realization of behaviors more powerful than probabilistic behaviors. The power of this new idea comes from the interference and entanglement; interference allows behaviors to create probabilistic distribution of outputs not possible

otherwise. Entanglement allows to not only transport information, but with an appropriate measurement procedure novel behaviors unique to quantum robot controller can be created. In particular, entanglement creates the constraint on robot's behavior. For instance if EPR circuit is used as a "quantum brain" of the above Braitenberg vehicle the entanglement allows for the above quantum Braitenberg behavior to discover the principle of going straight forward or straight backward ($\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$) but not turning left or right (this means that states $|01\rangle$ or $|10\rangle$ are never observed).

Figure 7.2: Circuit representing a robotic controller (with multiple robotic behaviors (labeled B_k)) with two input sensors and two motor control outputs. Each B_i is unitary matrix of certain elementary behavior.

Definition 7.1.1 Quantum Robot Behavior

Define the quantum robot as a set of behaviors $R = \{B_0, \dots, B_n\}$, where every behavior is a finite sequence of output states of the robot $B_i = \{b_0, \dots, b_k\}$. The size of the individual behaviors varies, allowing to express actions that are either single commands or full behavioral expressions. Thus one can write

$$R^q = \sum_{k=0}^l p_k B_k \quad (7.1)$$

and

$$B_k = \sum_{i=0}^{2^n} p_i |i\rangle \langle i| \quad (7.2)$$

where eq. 7.1 represents the selection of behaviors (task dependent) and eq. 7.2 represents each behavior as a quantum circuit (with p_i representing the probability of observing the state $|i\rangle \langle i|$). Thus a single behavior is equivalent with a quantum circuit described by a set of inputs and a set of corresponding measured outputs. This also entails the fact that as each behavior is defined on the same set of inputs and outputs (wires/qubits) the overall robotic behavior becomes a sequence of actions such that for each state $|\phi\rangle$ of the robot the output state has a finite probability to belong to behavior B_j (eq. 7.3).

$$\rho_k = \sum_{i=0}^l p_j \rho_l \quad (7.3)$$

An example of robot controller with four basis behaviors is shown in Figure 7.2 with each behavior being activated by a single input combination. $\square$

Example 7.1.0.1 Behavioral Sequence

Let's look at Figure 7.3. B_0 and B_1 are respectively the Toffoli gate and the Fredkin gate representing two behaviors applied on the outputs, the motor controls. A given observable behavior is now represented by the two possible behaviors that can be activated with a control input c allowing to switch between B_0 and B_1 behaviors that will generate a behavior given some inputs. For instance, for the set of sensory input states and control state (separated

Figure 7.3: Schematic diagram of circuit showing two behaviors controlled by a control bit. This bit can come from another input, the user of the robot or some higher-level quantum or classical controller.

by ";") $I = \{00; 0, 01; 1, 11; 0, 11; 1, 11; 0, 00; 0\}$ the corresponding behavioral sequence is given by $B = \{00_{B_0}, 10_{B_1}, 10_{B_0}, 11_{B_1}, 10_{B_0}, 00_{B_0}, \}$. Each generated state such as for instance 00_{B_0} represents the generated output state $|00\rangle$, obtained by applying the behavior B_0 (the control state is $|0\rangle$) on the motor qubits and using the sensory state as input. The diagram representing this robotic controller is shown in Figure 7.3.

□

Unlike in classical robot control, in quantum robots the output quantum states of the controller circuit must be measured. In general the measurement is subsumed on the outputs of the logic circuit as long as the desired output is represented in the initial basis. Thus for instance a permutative circuit assumes that the inputs and outputs are Boolean; both the inputs and outputs are defined on $\{|0\rangle, |1\rangle\}$. The observation of the outputs will be always generating a quantum-probabilistic distribution over the possible output states. However if one would measure for different output states, it is possible to introduce new control signals for the motors. For instance an entangled robotic behavior (such as defined by the Bell-Basis states), will generate a different behavior

which is not possible in classical computing and robotics. Also, according to the desired or expected output the learning algorithm can *invent* new output states for novel behaviors.

Note that Table 7.1 represents the controller as a Moore Finite State machine, the output depends only on the state. For instance, the input 00 changes the state to 11 and this has for consequence changing the outputs to 11. In the rest of this chapter, the K-map will be used for the design of robotic behaviors represented by Moore machines. For instance, it is possible to represent every quantum circuit as a 1QFSM (Appendix B, def. B.0.18). This is because every computation on a circuit is represented by three steps: Initialization, Computation and Measurement.

Each time one desires to compute a function for a set of inputs, the quantum register must be initialized in a given state, otherwise the quantum register will preserve its previous state. Moreover, we will assume that it is possible to initialize arbitrary number of qubits from a given quantum register in an arbitrary order. Thus for a three qubit quantum register one can initialize the register once, compute on the whole register and measure only a subset of qubits. All next computational steps initialize and measure only the input and output qubit while leaving the rest of the quantum state non-measured.

For instance, let $Q = \{q_0, q_1\} = \{|0\rangle, |1\rangle\}$ be two states of a single-qubit

quantum state machine M defined by the following Mealy equations:

$$\begin{aligned}
 U_0|q_0\rangle &\rightarrow |q_0\rangle|0\rangle \leftrightarrow |I_0, q_00\rangle \rightarrow |I_0, q_00\rangle \\
 U_0|q_1\rangle &\rightarrow |q_0\rangle|1\rangle \leftrightarrow |I_0, q_10\rangle \rightarrow |I_0, q_01\rangle \\
 U_1|q_0\rangle &\rightarrow |q_1\rangle|0\rangle \leftrightarrow |I_1, q_00\rangle \rightarrow |I_1, q_10\rangle \\
 U_1|q_1\rangle &\rightarrow |q_1\rangle|1\rangle \leftrightarrow |I_1, q_10\rangle \rightarrow |I_1, q_11\rangle
 \end{aligned} \tag{7.4}$$

where $|I_j, q_kO\rangle$ represents the input qubit, the state qubit and the output respectively. Observe that the matrices U_0 and U_1 are 4×4 as they act upon $|q_kO\rangle$, but both together they can be represented as a 8×8 quantum multiplexer controlled by I . Thus both state transitions U_0 and U_1 do not change the qubit I .

Equation 7.5 in the form of a Unitary matrix shows the defined state transitions from eq. 7.4 with input in the form $|I, QO\rangle$ (only the defined state transitions are showed below).

Figure 7.4: Schematic diagram of circuit showing two behaviors controlled by a control input qubit and with dedicated qubits for the outputs.

$$U = \begin{matrix} I_0 q_0, 0 \leftarrow \\ I_0 q_0, 1 \leftarrow \\ I_0 q_1, 0 \leftarrow \\ I_0 q_1, 1 \leftarrow \\ I_1 q_0, 0 \leftarrow \\ I_1 q_0, 1 \leftarrow \\ I_1 q_1, 0 \leftarrow \\ I_1 q_1, 1 \leftarrow \end{matrix} \left[\begin{matrix} U_0 = \begin{pmatrix} 1 & 0 & - & - \\ - & - & - & - \\ 0 & - & - & 1 \\ - & - & - & - \end{pmatrix} & \begin{pmatrix} - & 0 & - & - \\ - & - & - & - \\ 0 & - & - & 0 \\ - & - & - & - \end{pmatrix} \\ \begin{pmatrix} - & 0 & 0 & - \\ - & - & - & - \\ 0 & - & - & 0 \\ - & 0 & - & 0 \end{pmatrix} & U_1 = \begin{pmatrix} - & 0 & - & - \\ 1 & - & 0 & 0 \\ 0 & 0 & - & - \\ - & - & 1 & - \end{pmatrix} \end{matrix} \right] \quad (7.5)$$

Observe that we represent the FSM with a dedicated input $|I\rangle$, state $|q\rangle$ and output $|O\rangle$ qubits as in eq. 7.4 and we use an ancilla bit to represent the output by initializing it always as a constant $|0\rangle$. In the quantum robotic behaviors, we automatically assume that the behavior is applied to the dedicated output qubits. Thus a quantum behavior such as shown in Figure 7.3 can be represented as a circuit with five instead of three qubits (Figure 7.4).

Here the machine can be represented as a diagram and a circuit shown in

Figure 7.5, with three qubits, because the output does not correspond directly neither to the input nor to the state. The input to the machine and the output are represented as a ket vector above each state transition. For instance, the transition from state $|q_0\rangle$ to state $|q_1\rangle$ is triggered by the input $|1\rangle$ and generating output $|0\rangle$ (represented by $|1, 0\rangle$), while the transition from state $|q_1\rangle$ to $|q_0\rangle$ is triggered by input $|0\rangle$ and generates the output $|1\rangle$ (represented by $|0, 1\rangle$). Figure 7.5b shows the circuit representing the QFSM from Figure 7.5a. The I_0 block represents single initialization used only at the beginning of the computation. I_n is qubit initialization occurring at each computation step, and the M block represents the qubit to which the measurement is applied. Observe that in this case, only the output qubit $|O\rangle$ is measured while the internal state qubit $|q\rangle$ remains non-measured during the operation of the machine. It preserves the quantum state of the machine.

Figure 7.5: a) - The state diagram of the machine represented by the state transition function U_0 , b) - the schematic diagram of the circuit showing the initialization, the Unitary operator and the measurement on the output qubit.

Quantum FSM entails the possibility to generate superposed or entangled states in the machine. In the above example, we can modify the machine by altering its state transition by replacing U from eq. 7.5 with the new matrix

V.

$$V = \left(\begin{array}{c} U_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \quad U_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix} \right) \quad (7.6)$$

From eq. 7.6 the evolution of the machine is described by:

$$\begin{aligned} U_0|q_0\rangle &\rightarrow |q_0\rangle|0\rangle \leftrightarrow |I_0, q_0 0\rangle \rightarrow |I_0, q_0 0\rangle \\ U_0|q_1\rangle &\rightarrow \frac{1+i}{2}|I_0, q_0 1\rangle|1\rangle + \frac{1-i}{2}|I_0, q_1 0\rangle \\ U_1|q_0\rangle &\rightarrow |q_1\rangle|0\rangle \leftrightarrow |I_1, q_0 0\rangle \rightarrow |I_1, q_1 0\rangle \\ U_1|q_1\rangle &\rightarrow \frac{1-i}{2}|I_1, q_0 1\rangle|1\rangle + \frac{1+i}{2}|I_1, q_1 1\rangle \end{aligned} \quad (7.7)$$

In order for this specification to be a correct quantum machine the state qubit must be measured as well, which is shown in Figure 7.6. Observe that both the output and the state are measured and initialized at each computational step. By measuring, the complex superposition is destroyed and a basis state is obtained.

To completely explore the superposition in behaviors, one can look again back

Figure 7.6: Schematic diagram of the circuit showing the initialization, the Unitary operator and the measurement gates on both the output and the state qubits.

to Figure 7.2 but now allow for a superposition of control input qubits. The output is then the product of each activated behavior for particular state in the superposition. The output is represented as $U|o_0o_1\rangle = \Pi_{j=1}^{2^n} B_j \sum_{i,j \in \{0,1\}} |ij\rangle$. The above illustrates the point made in Chapter 3, that the synthesis of QFSM and FSM is equivalent to the quantum logic synthesis of combinational circuits of these machines.

7.2 The concept of learning robotic behaviors from examples.

Quantum Machine Learning has been already studied from various perspectives. Concept Learning, Template Matching, Machine Learning, has been studied as both an extension to classical machine learning as well as separate instance of novel learning methodologies for quantum computers.

One of the generalizations of the algorithmic approach are the Quantum Neural Networks and Quantum Neurons[Kak95, VM97, VM98]. This approach in general uses superposition for storing the weights of the neural network. Quantum Neuron generalizes the concept of the unit computation to use quantum computing to accelerate the learning and processing. Quantum Inspired Genetic Algorithms are also another generalization of computational

algorithms[NM96, HPLK01]. In this case the evolutionary operators can be represented as quantum operators operating on quantum strings of characters. In such a case one can put each individual in superposition and the application of unitary operators has a general effect on the superposition and thus allows faster learning and convergence.

From theoretical Machine Learning perspective one of the approaches is the formalization of the Concept/PAC Learning by Servedio [SG01, Ser01]. This approach was already extended to various classes of functions such as Juntas [Ati06, AS07]. Bshouty [BJ95] proposed a learning algorithm to learn DNF using oracle and generalizing the PAC learning.

The machine learning approach was also explored from the point of view of POVM matching. Sasaki proposed at first a probabilistic approach to estimate quantum state [SCJ01] and then extended it to the concept of the universal quantum matching machine [SC02].

Definition 7.2.1 Concept

Let set $C = \{c_0, c_1, \dots, c_m\}$ represents a class of concepts. Let $f(\mathbf{a}) = a' \in C$ and $a \in \{0, 1\}^{\otimes n}$ be a function mapping all Boolean inputs to C . Then f can be generalized to a concept if f accepts $a \in \{0, 1, -\}$ and $f(\mathbf{a}) = a' \in C$ if $\forall \{0, 1\} \in a$, all vectors $\mathbf{b}$ such that $f(\mathbf{b}) = b' \in C$ agree with $\mathbf{a}$ on all defined minterms.

For instance given a function $f(a, b, c)$ defined by the K-map

then the vector $\mathbf{v} = (1, -, 1)$ with $f(\mathbf{v}) = 1$, is a concept as both $f(1, 1, 1) = 1$

$ab \backslash c$	0	1
00	0	0
01	1	0
11	0	1
10	1	1

and $f(1, 0, 1) = 1$. In other words, a concept is a product of literals that is an implicant of function f (not necessarily a prime implicant of f) $\square$

The general idea behind Concept Learning is the use of inductive bias in order to satisfy the problem specifications, given the initial specification of the function. For instance, the ideal inductive bias is such that with each step in the learning procedure the learned representation exactly described the knowledge to learn. The default models discussed in the original paper are the DNF and the CNF Learning [Val84]. For clarity the CNF Learning is explained below.

Example 7.2.0.2 K-Literals CNF Learning

CNF or (Conjunctive Normal Form) is a product expression of n clauses c , with each clause being a sum of literals such as $\{l_0 + l_1 + \dots + l_k\}$. Thus $(p_0 + p_2)(p_3 + \overline{p_2} + p_4)$ is an instance of a CNF formula.

Let G^k be the set of all possible CNF's (products of clauses) with clauses of up to k literals in a clause. The procedure starts by querying Examples m -times, and for each obtained representative vector $\mathbf{v}$ of the Concept C , all clauses in $h \subset G^k$ with only negated or undetermined literals (denoted by $f_{h^{0,1}}^-$) that are determined in $\mathbf{v}$ are removed from G^k . This can be written as:

$$G^k = G^k \odot f_{h^{0,1}}^- \quad (7.8)$$

with $\odot(\cdot)$ represents the operation of removing all up-to k -literals clauses from the initial set G^k that do not agree with $\mathbf{v}$. This formulation can be written as a recursive operation. This is possible assuming a set of l distinct examples and the overall process of learning can be represented as in eq. 7.9.

$$G^k = (G^k \odot f_{h^{0,1}}^-)^l \quad (7.9)$$

□

There is a difference in the understanding of the word "*concept*" of ML and those of LS/QLS. While in ML the concept represents a generalized function class with the goal being the minimization of the number of queries to the Example/Oracle, in Logic Synthesis the goal is to realize a circuit from given functional constraints. In other words the PAC learning estimates how correct a model can be built from a given sample set (examples), while the LS attempts to build the optimal model representation with a given set of examples. This is illustrated in Figure 7.7a and Figure 7.7b. The PAC Learning uses a set of samples to estimate the correctness of the model (error δ).

Observe that the PAC learning (Figure 7.7a) uses Oracle and Examples (procedures containing the information about the original concept C) to build the target concept C' . The bottom loop including Examples, C' and δ represents

the learning of the concept C by querying Examples and modifying the target concept C' . The upper loop containing Oracle, δ and C' represents the learning of the target concept C' by querying the oracle on a user-generated example allowing to determine whether yes or no it belongs to the Concept we are learning.

Logic Synthesis (Figure 7.7b) uses a full set of specifications (all examples are available at once) to build a minimal logic realization of the desired function or concept. Moreover PAC learning considers the difficulty of designing the correct realization only with respect to the amount of knowledge. Thus in PAC learning both - completely specified and incompletely specified - functions are equally represented.

Finally, one can observe that logic synthesis methods applied to binary functions with many don't cares (don't knows) are used as a base of various machine learning (ML) approaches [ZBBD97, FP98, Fil98, Gry00]. While the method of logic synthesis based machine learning was already applied to binary and multiple-valued circuits [ZBBD97, Gry00], here it is applied for the first time to quantum circuits [NC00, MMD06, MS00] thus the learned behaviors include the entanglement which is a phenomenon not possible in the classical world.

With the above background we are able now to define the principles of inductive learning used in this dissertation.

Definition 7.2.2 Inductive Learning of completely Specified functions in Logic Synthesis

Figure 7.7: a) Schematic representation of Concept Learning with C representing the definition of the problem, C' representing the concept to learn. δ represents the error (difference between the original concept C and the target concept C'). The hexagon in the bottom left corner represents the operation of transforming the target concept C' by each query to the Examples (adding or removing minterms). The oval shaped object with (1||0) inside in the upper right corner represents the result of querying the Oracle with an generated example vector of C' resulting either in 1: the Oracle confirms that the given vector is an element of C , or in 0: the vector does not represent the concept class C . b) - representation of the Logic Synthesis process.

Let I be a set of vectors such that $I_k^p = \{i_0, i_1, \dots, i_n\}$, $n = 2^N$ is the k -th input vector (of N qubit) of pattern P (or function specification) and $f : I \rightarrow O$ be a reversible function, with $O_k^p = \{o_0, o_1, \dots, o_n\}$ being the expected result vector for the input pattern I_k^p and O is the set of all output vectors.

Let, $i_k \in \{0, 1\}$ and $o_k \in \{0, 1\}$ be the elements of the input and output vectors respectively and $\sum_{k=0}^{2^N} |\alpha|_k^2 = \sum_{r=0}^{2^N} |\alpha'|_k^2 = 1$ specifying the l_2 - norm space.

Let $|\psi\rangle$ be a 3-qubit quantum state and G be the set of possible operators (quantum gates).

Then there exists a quantum logic circuit U_f such that for any pair of input and output vectors (I_i^p, O_i^p) ; $I_i^p \in I^p$, $O_i^p \in O^p$ where $\forall O_i^p \in O \exists I_i^p \in I$ such that $f(I_i^p) = O_i^p$ is a one-to-one mapping. For quantum learning this means that there is a unitary transform on a quantum system $U_f|\psi\rangle \rightarrow |\psi'\rangle$ for $|\psi\rangle \in I^p$, $|\psi'\rangle \in O^p$. The learning of such a function implies to find the minimal set of quantum gates implementing function f as a Quantum Circuit (and realizing unitary matrix U_f). $\square$

The verification of the above definition is simple because the definition implies a permutative function mapping to which directly corresponds a single unitary transform (which is also permutative).

Example 7.2.0.3 Completely specified reversible function realized in quantum logic

Let f be a completely defined function represented in Table 7.2

Table 7.2: A K-map of a completely specified reversible function

C	0	1
AB		
00	000	001
01	011	010
11	100	101
10	110	111

One of possible realizations of function f is shown in Figure 7.8. This circuit can be found by QLS and represents the unitary transform U_f . The eight cells of the K-map from Table 7.2 correspond to eight input-output vectors (the learning examples). Thus input pattern $abc = |110\rangle$ is mapped for instance to the output pattern $PQR = |100\rangle$, etc. With respect to the definition 7.2.2

the obtained unitary matrix (quantum circuit) is a permutative matrix and is shown in eq. 7.10.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad (7.10)$$

As we see this function is reversible as it is a one-to-one mapping (set of output vectors is a permutation of the set of input vectors). Using matrix multiplication and Kronecker products of the elementary matrices of all 2×2 gates involved, one can verify that mappings of all cells (shown by the permutation of the inputs in Table 7.2) are satisfied [NC00, LPG⁺04]. The circuit in Figure 7.8 is thus the result of learning (synthesis) from the initial set of examples (Table 7.2). In this case there are many circuits to satisfy all input-output pairs, but they all have the same unitary matrix. $\square$

Although the Example 7.2.0.3 was given for the completeness of my presentation and to show the link between logic synthesis and learning, very rarely in real life the system learns (generalizes) from a complete specification.

Figure 7.8: Example of a completely specified 3×3 reversible function realized as a quantum circuit using quantum primitives CV , $CV^\dagger$ and $CNOT$ ($CNOT$).

Definition 7.2.3 Inductive Learning in Logic Synthesis for incompletely Specified functions

Let I be a set of input vectors defined as in def. 7.2.2 and let O be the set of output vectors such as in def. 7.2.2 but with $o_k \in [0, 1, -]$. The symbol '-' represents a don't care and corresponds to an unknown output. The set of examples is given as a set of pairs $P = \{i_k, o_k\}$, $k = 1, \dots, n \leq 2^N$.

For a given set of pairs (I_k^p, O_k^p) , the inductive learning for incompletely specified functions is the process of explicitly finding such a mapping or function that satisfies each pair (I_k^p, O_k^p) from the given set P such that $f(I_k^p) = O_k^p$. The result of learning is thus a circuit that describes a complete mapping that agrees with the set of input-output pairs from the specification examples. $\square$

Example 7.2.0.4 Synthesis of Quantum Circuit for Incompletely Specified Function.

Let f be a 3-qubit incompletely specified reversible function defined by the Table 7.3 (it can be checked that the function can be completed to a reversible map since all care cells are different). Table 7.3 represents thus the set P of learning examples called also the "problem specification". Then a valid solution to the learning problem specified in Table 7.3 is any unitary transfor-

Table 7.3: Incompletely specified reversible function f

C AB	0	1
00	000	001
01	-	-
11	100	101
10	-	-

mation U that satisfies all the specified transitions as in eq. 7.11

$$\begin{aligned}
 U|000\rangle &\rightarrow |000\rangle \\
 U|001\rangle &\rightarrow |001\rangle \\
 U|110\rangle &\rightarrow |100\rangle \\
 U|111\rangle &\rightarrow |101\rangle
 \end{aligned} \tag{7.11}$$

together with the corresponding circuit which behavior is specified by U . Thus the circuit from Figure 7.8 is a solution also to the learning problem specified in Table 7.3. Let us observe that in this case there are not only many circuits that solve this problem but there exist also many unitary matrices that solve this problem. Based on the Occam Razor Principle the circuit's complexity is reduced as to satisfy eq. 7.11 in the simplest possible way. As the byproduct of circuit's reduction the unitary matrix of this circuit is simplified as well. $\square$

To complete the definition of the incompletely specified learning, a closer look at the don't cares is given.

Theorem 7.2.4 Quantum Circuit for Incompletely Specified Learning

Any quantum-reversible incompletely specified function that:

- is built according to the above Inductive Learning Definition 7.2.3
- uses a quantum state $|\phi\rangle$ in a complex Hilbert space $H^{\otimes N}$ spanned by the set of basis states $|I\rangle$
- uses a set of single-qubit and two-qubit operators such as for example $G = \{I, CV, CV^\dagger, CNOT\}$

will result in a completely specified function allocating the unknown elements from the definition according to the unitary evolution matrix defined by the synthesized quantum circuit represented by a unitary matrix specified by:

$$U|I\rangle = |O\rangle \quad (7.12)$$

, where $|O\rangle$ is the binary basis output state vector. For every other state $I'\rangle$ there exists a unique quantum state $U|I'\rangle$. $\square$

To prove Theorem 7.2.4 is sufficient to observe that we select such gates that the unitary matrix of their compositions (using Kronecker product for parallel connections of gates and matrix product in reverse order for serial connections of gates) is a standard unitary matrix (with no don't cares). This matrix is created in such way that for every vector $|I_i\rangle$ from the pair (I_i, O_i) we have that $U|I_i\rangle = |O_i\rangle$. Applying matrix U to an arbitrary other input vector $|I'\rangle$, superposed or basis vector, produces certain output vector $|O'\rangle$ space (in

general a vector of complex numbers) so that $U|I'\rangle = |O'\rangle$. This vector $|O'\rangle$ is a completely specified quantum state in a sense that it is a quantum state that is known and deterministic (expressed by a wave equation). On the other hand, after the measurement, there may be very many classical states to which this state $|O'\rangle$ will collapse.

We see thus here a difference between the classical learning and the quantum learning. In classical learning we learn a deterministic function that maps inputs to outputs. In quantum learning we learn a quantum unitary mapping of input states to output quantum states. The output quantum states are deterministic only before a measurement. In this new type of learning the observer never knows to which classical state this deterministic quantum state will collapse as the result of the measurement. The designer of the robot sets thus certain constraints for robot's behavior but he can only probabilistically predict how the robot will behave within these constraints.

Taking into account the above introduced quantum phenomena, the concept of the quantum logic design and the inductive learning, the general mapping of the don't cares to cares in inductive learning can be classified into three categories. Figure 7.9 illustrates the three learning methods for single-output and multiple-output quantum functions. The quantum (or permutative) function to be learned is represented on the leftmost Karnaugh map (K-map) with cares (desired output values) and don't cares (that correspond to cases not known). To make this function reversible the input qubits are forwarded to outputs and a single ancilla bit is added (which is typical for quantum oracles [Gro96]) as

Figure 7.9: Three modes of learning based on the properties of quantum systems. On the left the desired function is represented using cares and don't cares. The top Karnaugh Map represents the result of the mapping being the deterministic learning. The bottom map represents probabilistic and the map on the right represents the learned function using Measurement dependent output.

in Grover algorithm.

1. The first type of learning is shown on the top of Figure 7.9 and it will be referred to as the deterministic learning (the classical learning of Boolean functions). The output of this deterministic learning is a completely defined Boolean function or a complete reversible Boolean function, if required (Example 7.2.0.4).
2. The second type of learning called Quantum-probabilistic is shown on the bottom of Figure 7.9. It has similar results as standard probabilistic learning, with the difference that the probabilities are obtained from quantum states (complex valued vectors) V_0 and V_1 (the values V_0 and V_1 will be explained later) according to the measurement operation.

3. Finally, the last type of learning is the measurement dependent (entangled) learning. It allows to specify the complete state of the system according to the selected measurement on a single qubit (this means that for observables on the single measured qubit the circuit will be performing different functions for the same inputs).

Figure 7.10: Schematic representing the possible learning methods in Quantum Computing. As defined in this dissertation, using QLS, the learning can be done for the data specified either before or after the measurement. Learning can be also done on a single or multiple qubits.

These methods are explained in details below and illustrated with examples:

1. A synthesis method based on symbolic quantum Karnaugh maps. In this approach we show how using heuristics and the synthesis process from a set of initial gates allows us to determine the circuit for the input-output mapping of the desired input-output pairs as well as mapping generated by the synthesis process for all don't cares.
2. A complementary synthesis method using measurement and entanglement to minimize circuits. This approach was developed in order to

minimize the number of measured qubits with respect to an expected result.

The inductive learning process presented here learns the entire description in one run (in contrast to incremental learning methods in NN and ML). Thus the Machine Learning is represented by the process of synthesis of the given function in a circuit with the minimal cost. This synthesis process preserves the cares but replaces all don't cares (don't knows) with deterministic, probabilistic or entangled states. The measured probabilities of outputs result from the circuit's structure and the types of controlled 2×2 gates used (here, we use gates that are controlled roots of unity with various angles).

The above function minimization with minimum cost of the circuit can be studied using a genetic algorithm (or exhaustive search for small number of variables or examples). The automatically synthesized controllers (i.e. learned from examples) produce very interesting robot behaviors [RFW⁺07, LP07]. Also, the synthesis method presented in this dissertation uses true quantum gates; unlike in other quantum logic synthesis problems here only single-qubit and two-qubit gates are used rather than abstract macros like a $k \times k$ Toffoli. Thus, the circuits learned by this method are directly implementable in quantum hardware and they satisfy Occam Razor with respect to the real hardware costs. Then, we can say that these behaviors are more natural to quantum world than behaviors learned with many-input Toffoli gates that would push the solutions towards less probabilistic behaviors or to no probabilistic behaviors at all. This is again more similar to human learning that includes

always some probabilistic component (at least as related to body motions and to speech).

7.2.1 Symbolic Quantum Synthesis of Single Output Quantum Circuits

Assume a single output function is to be synthesized and is defined by the Karnaugh map. This K-map specifies the probabilities of observing values 0 and 1 as well as the output states and is shown in Table 7.4.

Table 7.4: Single output Boolean functions R to be synthesized (learned)

c	0	1
ab		
00	-	-
01	0	-
11	1	0
10	-	1

R_0

The Table 7.4 (R_0) has half don't cares and half cares, and assuming there is a method to synthesize the cares, the problem that remains to be solved is in what manner it is possible to specify the don't cares in order to minimize the overall cost of the solution circuit. The unitary operators used in this work are [W] (Wire/Identity), [NOT], [V], $[V^\dagger]$, [CNOT], [CV] and $[CV^\dagger]$ and thus it should be specified how the don't cares are filled with respect to inductive learning of the robotic behaviors.

In general in ML the specification of a given Concept to find, the Oracle (that has a direct counter part in quantum computing) defines a function $F(s) = 1$. From an objective point of view the structure of the concept itself is not the main goal as long as the learned concept/function satisfies the specifications.

In Logic synthesis the structure of the circuit is directly responsible of the inductive bias in the process of transforming the initial incompletely specified function to the final one. Thus the synthesis of quantum circuits in Logic Synthesis directly influences the process of generating quantum states with inductive bias directly related to the used gate primitives. This means that while using only permutative gates the output will be always a permutative function, using other gates such as CV the output of the circuit will be a probabilistic function with output states having the observation probabilities directly dependent on the used component gates. Incompletely specified function f synthesized into a circuit representing function f' containing only states $\{|0\rangle, |1\rangle\}$ can be much more costly to build than one that would only generate such states for the desired values and would fill the remaining don't cares with possibly quantum probabilistic or even entangled states.

Definition 7.2.5 Symbolic Quantum States

Let U be a unitary operator representing a Quantum Circuit with single qubit outputs $O = \{o_0, o_1, \dots, o_k\}$ with $k < 2^n$. One can describe the observed outputs with a set of such observable symbolic values $S = \{s_0, s_1, \dots, s_p\}$ that each output o_n can be represented by a symbolic state s_j . $\square$

Observe that for single-output quantum functions the states from S can take any possible form described by $S \rightarrow \alpha|0\rangle + \beta|1\rangle$ and represented by designing the appropriate measurement to capture the desired quantum state. For multi-output quantum function, the symbolic states can take the form of any multi-qubit quantum state such as multi-qubit superposition or entanglement.

For this, let $S_o = \{0, 1, -, V_0, V_1\}$ be the set of all possible (symbolic) outputs of a given single output function (qubit R in Fig 7.11). Then, 0 and 1 represent 100% of obtaining 0 and 1 respectively after observing (i.e. measuring) the system's output. $V_0 = V|0\rangle$ and $V_1 = V|1\rangle$ are symbols that represent quantum states (vectors of complex numbers). To these quantum states correspond, by measuring/observing the system, the states $M(V_0)$ and $M(V_1)$ with $M = \{M_{V_0}, M_{V_1}\}$ being the measurement operators (eq. 7.13).

$$M_{V_0} = |V_0\rangle\langle V_0|, \quad M_{V_1} = |V_1\rangle\langle V_1| \quad (7.13)$$

Example 7.2.1.1 Quantum Symbolic Synthesis I

It is possible to synthesize any single output function using the above set of gates with the expected value of the outcome specified by (set of states) S_o defined on this page above. An algorithm for this problem has been already shown in other works [HSY⁺04] however the important point is the methodology. A systematic application of the above gates with respect to the desired qubit will realize this function. Illustration of this method for reversible multi-output functions was given in [HSY⁺04]. The above function was realized using the GAEX algorithm presented in chapter three and four. The incompletely specified function from Table 7.4 specified for a single output variable takes now the following form: $f = [0, 1, 0, 1, 0, 1, 1, 0]$ replacing all don't cares with either 0 or 1.

Below is an example of an obtained result using automated symbolic synthesis

(using the $CV/CV^\dagger$ gates). The circuit is shown in Figure 7.11 and as it can be seen in Table 7.5 the behavior of this circuit is fully deterministic.

Figure 7.11: Example of a solution to the learning problem for the incomplete function from Table 7.4. This circuit has a deterministic behavior.

Let us note that when all controls of gates are binary the possible quantum states are only 0,1, V_0 and V_1 (this was proven in [HSY⁺06]). Observe that the overall methodology is to analyze the application of NOT and V gates on the target qubit. The method of doing this is to apply single-qubit controlled single-target quantum gates or single qubit-controlled gates controlled by affine functions (constructed using CNOT gates) on the target qubit. This can be observed in more details on the implementation from Figure 7.11 and in Table 7.5.

Table 7.5: Example of analysis on circuit from Figure 7.11

$ab\ c$	0	1		$ab\ c$	0	1
00	I_a, I_b, I_c	I_a, I_b, I_c	→	00	I_a, I_b, I_c	I_a, I_b, I_c
01	$I_a, I_b, (VV^\dagger)_c$	$I_a, I_b, (VV^\dagger)_c$		01	I_a, I_b, I_c	I_a, I_b, I_c
11	$I_a, NOT_b, (VV^\dagger)_c$	$I_a, NOT_b, (VV^\dagger)_c$		11	I_a, NOT_b, I_c	I_a, NOT_b, I_c
10	$I_a, NOT_b, (V^\dagger V^\dagger)_c$	$I_a, NOT_b, (V^\dagger V^\dagger)_c$		10	I_a, NOT_b, NOT_c	I_a, NOT_b, NOT_c

$ab\ c$	0	1
00	$ 000\rangle$	$ 001\rangle$
01	$ 010\rangle$	$ 011\rangle$
11	$ 100\rangle$	$ 101\rangle$
10	$ 111\rangle$	$ 110\rangle$

The first (top-left) Table 7.5 can be formally minimized by analyzing output on each qubit (for functions with multiple outputs, the above methodology

applies as well but the analysis is more complicated). $\square$

One could however desire to restrict the specification of a gate or a circuit by quantum related symbolic values. Using the above set of gates and the input states, it is easy to show that all the possible states on the circuit output are from the set of the states would be $\{0, 1, V_0, V_1\}$. Thus the output state probabilities directly depend on the logic elements (quantum gates) of the synthesis. If for example we would extend the set of gates to include also the gates $[\sqrt{v}]$, $[\sqrt{v^\dagger}]$, $[\sqrt[4]{v}]$ and $[\sqrt[4]{v^\dagger}]$, the output value probabilities would be appropriately changed.

Example 7.2.1.2 Quantum Symbolic Synthesis II

A more interesting case is shown in Figure 7.12, representing the result of automated synthesis and minimization of the Toffoli gate, specified with the output in only a single qubit (i.e. we are interested only in the Davio expansion $ab \oplus c$ part of the Toffoli gate).

Figure 7.12: Example of learning from a set of example pairs corresponding to one output of a Toffoli gate ($ab \oplus c$). Observe that when the control outputs are not forced to be restored, the overall cost of the gate is smaller.

The circuit from Figure 7.12 does generate deterministic output for the single measured qubit $|c\rangle$. However, it has probabilistic behavior for multiple qubit

Table 7.6: Example of analysis on circuit from Figure 7.12

$ab\ c$	0	1		$ab\ c$	0	1
00	I_a, I_b, I_c	I_a, I_b, I_c	$\rightarrow$	00	I_a, I_b, I_c	I_a, I_b, I_c
01	$V_a^\dagger, I_b, (VV^\dagger)_c$	$V_a^\dagger, I_b, (VV^\dagger)_c$		01	$V_a^\dagger, I_b, I_c$	$V_a^\dagger, I_b, I_c$
11	$I_a, NOT_b, (VV)_c$	$I_a, NOT_b, (VV)_c$		11	I_a, NOT_b, NOT_c	I_a, NOT_b, NOT_c
10	$V_a^\dagger, NOT_b, (VV^\dagger)_c$	$V_a^\dagger, NOT_b, (VV^\dagger)_c$		10	$V_a^\dagger, NOT_b, I_c$	$V_a^\dagger, NOT_b, I_c$

$ab\ c$	0	1
00	$ 000\rangle$	$ 001\rangle$
01	$ V_110\rangle$	$ V_011\rangle$
11	$ 101\rangle$	$ 100\rangle$
10	$ V_110\rangle$	$ V_011\rangle$

measurement. This can be seen in Table 7.6. For example the input state $|100\rangle$ can be written as:

$$\begin{aligned}
|100\rangle &\xrightarrow{U} \\
&\xrightarrow{W, CV} |100\rangle \\
&\xrightarrow{CNOT, W} |110\rangle \\
&\xrightarrow{CV} |11\rangle \frac{1+i}{\sqrt{2}} (|0\rangle + i|1\rangle) \\
&\xrightarrow{V^\dagger C, W} \frac{1+i}{\sqrt{2}} (|0\rangle + i|1\rangle) |1\rangle \frac{1+i}{\sqrt{2}} (|0\rangle + i|1\rangle) \\
&\xrightarrow{W, CV^\dagger} \frac{1+i}{\sqrt{2}} (|0\rangle + i|1\rangle) |10\rangle
\end{aligned} \tag{7.14}$$

Thus the original three-to-one incompletely specified function in Table 7.4 is mapped to the permutative reversible quantum 3×1 and to a quantum probabilistic 3×3 function by the circuit from Figure 7.12. Observe, that this case is more general as the circuit in Figure 7.11, because the presented method allows also complex control signals.

□

7.3 Experiments and Results for learning Quantum Behaviors

7.3.1 Symbolic synthesis - Single output functions

The GA was used to search three and four qubit-single output circuits for incompletely defined functions. Some of the benchmarks have already been introduced in Chapter 7 and they all are incompletely specified functions (sets of learning examples). The Table 7.7 shows the benchmark functions used in this chapter.

Table 7.7: GA benchmark functions

$f_1 = [-, -, 0, 1, 0, 1, 1, 0, 0, 0, -, -, 0, 1, 1, 0]$	4
$f_2 = [0, 1, -, -, -, -, 1, 0]$	3
$f_3 = [0, 0, -, -, 0, 1, V_0, -]$	3
$f_4 = [0, -, 0, -, V_0, 1, -, 1]$	3
$f_5 = [-, V_0, -, 1, -, 1, V_0, -]$	3
$f_6 = [0, 0, 0, 1, 0, 1, 1, 1]$	3
$f_7 = [0, -, 0, M_{0,1}, M_{0,1}, 1, -, 1]$	3

For the discussion of the results and synthesis four of these benchmarks have been selected below. Functions f_4 , f_5 and the Majority gate are analyzed and studied in details.

Function f_4

The first function was $f_4(a, b, c) = M_f = \{M_0, -, M_0, -, M_{V_0}, M_1, -, M_1\}$ with the least significant qubit being the output. The M_{V_0} represents the fact of expecting a probabilistic behavior. The realization is shown in Figure 7.13, with top bit being constant 0 $|0\rangle$ and bottom bit being the output. Because

	$c=0$	$c=1$
00	0	—
01	0	—
11	—	1
10	V_0	1

The next benchmark is another partially specified function f_5 , shown in Figure 7.14. Similarly to the previous example, this function is defined over a set of learning examples being a subset of possible mappings. Circuit from the Figure 7.14 generates the desired output states such as: $\{|001\rangle \rightarrow \frac{1}{\sqrt{2}}(|000\rangle + |010\rangle + |100\rangle + |110\rangle) + \frac{1}{\sqrt{2}}(|101\rangle + |111\rangle), |011\rangle \rightarrow \frac{1}{\sqrt{2}}(|101\rangle) + \frac{1}{\sqrt{2}}(|111\rangle), |111\rangle \rightarrow \frac{1}{\sqrt{2}}(|001\rangle) + \frac{1}{\sqrt{2}}(|011\rangle), |100\rangle \rightarrow \frac{1}{\sqrt{2}}(|000\rangle + |010\rangle + |100\rangle + |110\rangle) + \frac{1}{\sqrt{2}}(|001\rangle + |011\rangle)\}$. As can be seen this function uses superposition to generate the correct result.

Figure 7.14: Benchmark function $f_5(a, b, c)$.

Interestingly, adding two gates: $CV^\dagger$ on wires b and c at the beginning of the circuit and CV right after the first controlled-Not on the a and b wires, will generate partially deterministic and partially probabilistic behavior. Again, it is possible to describe this function using quantum symbolic values: $[0, V_1, 0, 1, V_0, V_1, 0, 1]$.

Majority Gate

The last result is for the 3-qubit majority function [MMD06] $M_m = \{M_0, M_0, M_0, M_1, M_0, M_1, M_1, M_1\}$. The function is shown on Figure 7.15 and is a pure permutative matrix.

Figure 7.15: Function 3×1 Majority.

In this case the analysis is a little more complicated $[VV^\dagger, V^\dagger V^\dagger, VV^\dagger, VV^\dagger, V^\dagger V^\dagger, VV^\dagger V^\dagger V^\dagger, V^\dagger V^\dagger, VV^\dagger]$ however again the correct result is obtained. Observe that this gate illustrate how gates $[Y]$ and $[V]$ interact to create deterministic reversible quantum circuit. For illustration the matrix representing

this circuit is shown in eq. 7.15. In particular observe that the realized function is deterministic but exactly half of the non null output coefficients are purely complex and the other half is purely real.

$$U = \begin{pmatrix} 0 - 1i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 - 1i \\ 0 + 0i & 0 - 1i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 - 1i & 0 + 0i \\ 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 1 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ 0 + 0i & 0 + 0i & 0 + 0i & 1 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ 0 + 0i & 0 + 0i & -1 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & 0 + 0i & -1 + 0i & 0 + 0i & 0 + 0i \end{pmatrix} \quad (7.15)$$

7.4 Discussion on learning Quantum Benchmarks

So far in this chapter, we showed a new alternative approach to machine learning; i.e. learning quantum circuits from partially specified examples with symbolic quantum states. This has application in robotics [RFW⁺07, LP07] (Appendix C). The behavior is specified by a truth table with don't cares and symbolic quantum states such as $V_0 = V|0\rangle$ and $V_1 = V|1\rangle$ which lead to known probabilities of observation of the output under certain measurement operators[LP05a]. A circuit matching the example symbolic cares is found

that has deterministic, probabilistic or entangled behaviors when measured. Such behaviors have been practically observed by us on several small mobile and humanoid robots[RFW⁺07, LP07]. This chapter introduced also the concept of a quantum function with binary inputs and single (quantum symbolic) binary output as well as the concept of circuits that use the measurement operator as one more component in the quantum logic synthesis method. We formulate the synthesis problems for these types of functions and solve these problems using genetic algorithm and exhaustive search. Quantum truth tables can be represented as standard Karnaugh maps with symbolic quantum states as their outputs. This new representation has a didactic value because it links the quantum concepts to the binary concepts such as minterms, symmetry, unatness, implicants and other that researchers and engineers are familiar with while using K-maps. This representation allows to create new methods to synthesize quantum circuits from K-maps that are more knowledge-based than the GA or exhaustive search methods. Also, a more appropriate notation for quantum circuits (with respect to measurement) was shown and used as the error evaluation in the Genetic Algorithm. Finally, the circuit synthesis using measurement operators and the change of quantum computational basis was introduced and explained. This approach allows to realize permutative boolean functions using entangled circuits and a well selected set of measurement operators allowing to collapse the entangled circuits into real observable unique basis.

The genetic algorithm implementation is extensible from single to multiple

qubits as outputs, by specifying the location of the measurement operation. The probabilities of result being in certain states are proportional to the desired states. It is achieved, however, only for classical probabilities and not for complex probabilities (quantum amplitudes).

Consider the point of view of Machine Learning formulated with respect to the formulated above criteria with learning the circuit's structure including the measurement operators. The problem of synthesizing quantum robotic behaviors can be looked at with these assumptions as virtually breaking the model into two separate parts. Figure 7.16 schematically describes the concept of the synthesis method of the logic block before the measurement (Learning 1) and after measurement (Learning 2). The learning process before the measurement is based on the evaluation of the unitary evolution without requiring explicit usage of measurement. This is the case when synthesizing for boolean logic for example; a circuit that generates deterministic outputs does not need to assume the usage of measurement (despite the fact that there must be measurement) because the outputs are collapsed into the observable basis of the quantum system. The second type of learning, requires the measurement because the final state of the system (and the output) might not be in some observable state.

This means that the synthesis process applies systematically gates and uses minimization technique to evaluate the output with respect to the unitary matrix representing the unmeasured state of the circuit. The behavior evaluation, however comes after the measurement because the control signals of the robot

must be classical binary.

Figure 7.16: Explanation of two models of quantum learning: Learning 1 represents the method that approximates complex coefficients as represented by the quantum states (vectors of Hilbert space) in the quantum circuit. Learning 2 represents the learning of binary vectors after the measurement (observable). This second method evaluates the learning result (the circuit) only on its observable behavior.

From the point of view of synthesis it is valid to ask if either the superposition or the entanglement can be used as tools for logic synthesis. In other words, can the given function be designed more efficiently while using superposition and entanglement as the resources? So far, the quantum logic synthesis research has been on the usage of a particular set of gates for minimization, taking into account entanglement that occurs in the synthesized circuits. But it is possible that future logic synthesis methods will use entanglement, superposition and measurement to synthesize desired functions, in either measurement-independent or measurement-dependent ways. This dissertation is the first step in this direction.

The problem of using the measurement in synthesis results however from the fact that in order to create a logic circuit using the measurement as a parameter the expected result must be stored. Thus, the observation of the output

value is controlled by the desired pre-stored output. This also means that for different desired quantum states on the output, one must prepare different measurements.

Finally, the future work includes extension of the synthesis process to multi-output and multi-valued reversible and truly quantum functions and software simulation of the quantum controllers on physical robots. In particular, we are investigating the quantum robot controllers with respect to social behavior and the automatic generation of social behaviors. An illustrative example of such application is described in Appendix C.

7.5 *Measurement Synthesis*

The measurement operation can affect only the measured qubit, a subgroup of qubits or all the qubits in the circuit (robotic controller). In the model of robot used in this dissertation (Appendix C), the quantum controller is embodied in real physical body. The body part initializes the quantum register and applies unitary transform on the quantum system and measures the output in a clocked manner (Figure 7.17).

As illustrated in the previous subsection, when dealing with multi-input multi-output quantum functions it is possible to minimize the expression. However such qubit based method is not well suited when the minimization of the circuit is done with respect to all physical constraints (initialization, measurement, etc). As will be seen this is also the case for functions implemented

Figure 7.17: The schematic of a quantum controller used in a classical robot. All operations are synchronized by one or multiple clocks assuring the correct outputs are generated for the appropriate inputs. Blocks S and O represent the classical sensors and actuators wires, I and M represent the initialization and the measurement operations of the quantum controller and U is the actual operation (unitary transformation) applied on the quantum circuit.

in a measurement dependent manner. To explain, in quantum computing everything is an active operation (the act of doing nothing still represents the identity operator application) and the measurement is a non-unitary transformation that requires explicit control operation from the non-quantum part of the robotic controller. Thus, one can minimize the synthesis process with respect to the measurement. In other words, the learning problem here becomes: "how to synthesize desired quantum behavior with a single single-qubit measurement?" Naturally, one approach consist in synthesizing deterministic (permutative) quantum circuits, but this approach is not discussed here as it does not improve the already known methodologies.

The approach to QLS described in this section uses entanglement as a source of output logic states. Using single-qubit measurement the output binary-state is obtained and compared with pre-stored desired output state. If the measured state matches the desired state, the computation is considered to be successful. Otherwise it is re-done, until matching result is obtained. The

general idea is that the selection of the measurement output value determines the implemented function. As the entanglement provides two possible outputs for each input, this approach can have potential usefulness for such problems where the overall output can be generated faster by a single measurement and a classical LUT is used to store the output values of the whole computation.

Definition 7.5.1 Entanglement operator

Let $|\psi\rangle$ be quantum register of length n in an initial state $|00\dots 0\rangle$. Then

$$\Sigma|00\dots 0\rangle = \prod_{k=0}^{n-2} \sigma_k |0_k 0_{k+1}\rangle \sigma_{k+1} |0_{k+1} 0_{k+2}\rangle \dots \sigma_{n-2} |0_{n-2} 0_{n-1}\rangle \quad (7.16)$$

creates a maximal entangled state in the Bell basis when the σ_w represents the entanglement operator applied to qubits. Observe that the implementation of the EPR operator on large scale can be executed in three steps:

- - initialize the system in state $|00\dots 0\rangle$ and apply Hadamard gate on the topmost qubit in the system (or a selected subsystem),
- - chain down the CNOT gates
- - apply NOT gates on desired qubits to obtain the desired input-output mapping (if different than obtained from state $|00\dots 0\rangle$). For example, a three qubit system initialized to $|000\rangle \xrightarrow{\sigma} \frac{|000\rangle + |111\rangle}{\sqrt{2}}$, by applying the NOT operator to the top-most qubit, the generated output will be $|000\rangle \xrightarrow{\sigma} \frac{|011\rangle + |100\rangle}{\sqrt{2}}$. This is because any unitary operation is a rotation in space and thus affects both parts of the quantum state in the superposition.

Consequently, it can be represented as a hypercube translation (Figure 7.18).

Figure 7.18: The application of the NOT gate to the entangled system creates rotations of pairs of vertices (minterms, K-map cells) within the hypercube.

□

Theorem 7.5.2 Entanglement Quantum Circuit

A quantum system that undergoes the evolution using the σ (def. 7.5.1) operator is fully entangled, and knowing the structure of σ (def. 7.5.1) the whole state of the system can be known using a single measurement on a preselected qubit (expected value). □

Proof. As each quantum state of a maximally entangled system is mapped to mutually negated logic pairs (such as $000 + 111$ or $010 + 101$), each measurement on the output qubit will project the system to one of two possible basis states. This property is illustrated in Figure 7.19: when the outcome of the measurement is $|0\rangle\langle 0|$ the state of the whole register is $|0110\rangle$ while when the output of the measurement is $|1\rangle\langle 1|$ the state is $|1001\rangle$. □

Figure 7.19: By pre-storing in additional non-quantum circuit the possible outcomes of a single qubit $|d\rangle$, the desired function can be realized using the entanglement operation. With measurement outcome $m = 0$ on qubit $|d\rangle$ the state is $|abcd\rangle = |0110\rangle$ and with measurement outcome $m = 1$ the state is $|1001\rangle$. Thus with single measurement 0 the control state 0110 is given to the actuators of the robot.

Because the output of the measurement of an entangled quantum state is still a quantum coin toss, the result of the measurement requires to be compared to an *expected output* on the measured qubit. Such setup requires an external classical memory such as ROM (a LUT) and can be seen on Figure 7.20b. Observe that the ROM stores classical binary signals that are addressed using the input vector and the content is the value selecting what measurement should be used. The measurement determines the whole state of the resulting quantum computation.

For the synthesis of a single output function, the whole state might be irrelevant to the desired output of the designed function. However in the case of a robot we require that all of the outputs must drive well-formed signals in order to properly control the actuators (Figure 7.20). In robotics, where control signals must be binary (as the actuators are using classical control signals) the output from the quantum controller must be translated from measured quantum signals to classical binary signals.

Figure 7.20: Schema representing the setting of measurement dependent computation for robotic control. (a) - a quantum circuit implementing the entanglement, (b) - the LUT storing the control signals for the robot and allowing to associate single measurement output with appropriate multi-qubit control signal and (c) the classical wires for the actuator control. The binary signals of the actuator wires are a function of inputs, of the measurement in qubit $|c\rangle$ and of the logic function stored in ROM. Observe, that the addressing mechanism is using classical inputs from the sensors; the quantum circuit is initialized only after the address has been assigned in order to avoid entanglement between the quantum circuit and the addressing mechanism.

Definition 7.5.3 Measurement Dependent Quantum Logic Circuit

We define the Measurement Dependent Quantum Logic Circuit (MDQC) having the following components:

- An entanglement circuit (Figure 7.21a)
- An expected outcome LUT/ROM (Figure 7.21b)

The computation procedure is defined as follows:

- Initialize the quantum array using an appropriate operator σ (definition

Figure 7.21: Entangled circuit representing the mechanism of selecting the desired function. The operator M_n is the conditional operator that allows to select correct expected measured values. The choice of the operator M_n is decided by the ROM at right. In this case, the addressing mechanism is using measured quantum inputs from the sensors; the quantum circuit is initialized only after the address has been assigned in order to avoid entanglement between the quantum circuit and the addressing mechanism.

7.5.1)

- Add an additional affine function as pre or post-processor to the circuit
- For each input state do:
 1. propagate the input through the circuit
 2. measure single qubit to obtain a measurement result
 3. compare the obtained output value of the circuit to the appropriate stored value in the LUT (Figure 7.21c)
 4. if match is obtained, use LUT to obtain the multi-bit signal and propagate it through the classical control logic to the robot's actuators
 5. if no match is obtained return to point 1 and redo the procedure

again.

Figure 7.21 represents the concept of the measurement-dependent logic circuit. The 2×2 entanglement operator $\sigma = ([H] \otimes [W]) \star CNOT$ constructed as described in definition 7.5.1 is given in the middle of Figure 7.21, indicated by the dotted line b). The rightmost logical element is the measurement conditional and the desired output register (dotted brace c). The leftmost part of the circuit represents the fact that the input is used as an address in the expected output register.

The addressing mechanism works as follows: for a given input state $I \in \{0, 1\}^2$, the output qubit (qubit b from Figure 7.21) is compared to the expected result value from the ROM. The result is that a particular measurement operator M_n is selected as M_0 or M_1 depending on the number stored in the respective address of the ROM.

Another example of such input-dependent measurement selection is shown in Figure 7.22. Observe that in the Figure 7.22 the signals controlling the mea-

Figure 7.22: The Measurement-dependent Synthesis: inputs are measured and the observable values determine which measurement is to be applied to the target qubits. This mechanism is also used to select the pre-stored output states in the MDQC

surement operation are measured quantum signals, while in the previous case the measurement is controlled by the classical signals from the robotic sensors Figure 7.20 . Also in Figure 7.21, the quantum signals are first measured and then the quantum circuit is re-initialized. These different methods of using the measurement for computation allow more or less freedom in synthesis process.

□

Using this approach, one two-qubit entangled circuit can represent up to four functions. For instance, the circuit from Figure 7.21 can represent up to four reversible 2×2 permutative logic functions. These functions are: $f_0 = [00, 01, 11, 10]$, $f_1 = [00, 10, 11, 01]$, $f_2 = [11, 01, 00, 10]$, $f_3 = [11, 10, 00, 01]$.

This can be explained as follows: assume there is a 2×2 entanglement circuit. In such case one can measure a single qubit for two boolean values for each input/output pair. Because the overall result should be a reversible function, the number of measurements for the value 1 and for the value 0 must be the same. Thus, for a given 2×2 entanglement circuit there are two measurements for 0 and two measurements for the value 1 on a single qubit. This means that there are 4 possible configurations of a single qubit measurement preserving the output as a function. These single qubit measurements are given by: $[0, 1, 1, 0]$, $[0, 0, 1, 1]$, $[1, 1, 0, 0]$ and $[1, 0, 0, 1]$ and are corresponding to the function f_0 , f_1 , f_2 and f_3 respectively.

The measurement is an external operation (to the quantum system). A physical device, external to the measured quantum system, is necessary for the

measurement in a robot controller that includes a quantum computer. The classical part of the robot is executing and controlling all the operations on the quantum system. Thus each of such actions should be included when one is evaluating for the cost of running such a computer. As such, a quantum computer can be minimized with respect to logic, technology, and also with respect to the cost of the measurement operations

In the case of robotic behaviors, the quantum computer can be used to quickly select information from a database of behaviors using a single qubit measurement. An input-output expected outcome LUT is needed for this, as it was explained in definition 7.5.3. In the present case the idea of such a robot is to allow to control the robot by a single qubit while keeping the rest of its controller unobserved. This could be seen as a robot that creates its actions based on calculations made in a quantum world and that executes these actions in the classical macro world.

Moreover, this approach is well situated for building quantum state machines where only a subset of inputs are initialized while the machine's internal state is left unobserved and evolves without measurement.

From the point of view of robot behaviors, the properties of the quantum system imply the following laws:

- The robot behavior is composed of Hilbert space states that have both real and imaginary components, thus the behavioral expression is observable up to the magnitude component of the states. The phase infor-

mation is lost in the measurement process. For example two states such as $|1\rangle$ and $|i\rangle$ are not recognizable because $|1\rangle\langle 1| = 1$ and $|i\rangle\langle i| = 1$.

- The expectation of a certain value associated with the measurement will in particular cases affect the outcome of the behavior.
- The application of the individual transforms in a given behavior, might not be always observable but affects the system at least on the imaginary plane. These accumulated changes of phase may affect future behaviors of the robot.

For learning such behaviors, it is necessary that that not only the behavior is tested for its correctness but it must be also checked if the behavior is or is not measurement dependent. Because of quantum effects, the observation (i.e. the measurement operation) plays often a role of the determination of the result.

There are four cases in designing/learning robotic behaviors with respect to inductive learning that are of our interest:

- In the case of a specification of the behavior by a Boolean function, the input-output is a one-to-one mapping.
- When don't cares are included in the function/behavior definition, then the following types of quantum behaviors can occur:
 1. The result can be a superposed state. The output behavior is probabilistic and the output of the circuit is independent from the type

of the measurement. In another words the measurement projects the system onto basis states with real-valued probabilities and the expectation values of m_0 and m_1 that respect the unity equation $m_0 + m_1 = 1$. However, to determine the total state of the system, (1) the measurement must be performed on every single qubit; (2) the system must be observed as a whole, and (3) the final state is determined by $|\psi'\rangle = M_n^{\otimes k}|\psi\rangle$.

2. The result can be a completely defined state.
3. The result can be a measurement-output dependent state. This is the case of entangled system, such as $|01\rangle \rightarrow \frac{|00\rangle+|11\rangle}{\sqrt{2}}$, the output will be determined by whether the output is observed for 0 or for 1. The measurement, collapsing the measured qubits to the projecting base, determines the output of the whole system. Thus for a system with more than one qubit in a fully entangled state, the output can be specified through $|\psi'\rangle = M_n|\psi\rangle$.

Example 7.5.0.1 Quantum Behaviors

Let us assume the robot controller from Figure 7.2. With respect to the possible values as the input signals from the sensors, the possible robot behavior is either B_0 , B_1 , B_2 or B_3 exclusively as the qubits controlling these behaviors take only values $|0\rangle$ or $|1\rangle$ because the input from the classical sensors are binary values. Figure 7.23 shows that when the selection of the robotic behaviors is based on a qubit dedicated to the robot quantum state that is not required to be measured, the output is a probabilistic mixture of all possible

Figure 7.23: Schematic diagram of a quantum robot in a real environment. The Controls qubit are not measured and in combination with the discrete Sensor inputs they can generate a mixture of behaviors on the output as well as many quantum internal states.

behaviors. Observe, that whether or not the final behavior is measurement-dependent depends only on the entanglement of the output state generated by the robot. Moreover, a measurement-dependent behavior can be generated both by a dedicated entanglement operator σ or by a combination of behaviors controlled by the superposition of the control qubit on the target qubits. $\square$

Assume the following two-qubit reversible function

$$\begin{aligned} f &= \{00 \rightarrow 01, 01 \rightarrow 00, 10 \rightarrow 11, 11 \rightarrow 10\} \\ &= [01, 00, 11, 10] \end{aligned} \tag{7.17}$$

With respect to the previous considerations, this function can be either built as a Boolean one-to-one mapping such that the result is completely defined and

100% identical to the original $f = [01, 00, 11, 10]$ (this mapping is specified by the permutation matrix) or in order to capture all features of quantum computing this function can be implemented as a circuit with measurement operators. This means that we are looking for a notation that will allow to represent a given behavior as a function of measurement in a unique way.

Definition 7.5.4 Measurement-Dependent Observable

Define $m_n = (M_n = 1)$ as the result of the measurement for value n on the output qubit after the computation using entanglement operator σ has terminated and after applying the measurement operator M . For $m_n = 1$, the system has been observed in state n and projected by M_n . Then the whole system state represented by $|\psi'\rangle$ must be a unique state corresponding to $|\psi'\rangle = M_n|\psi\rangle$, with $|\psi\rangle$ being the output state of the quantum system. $\square$

Using this definition, the equation 7.17 can be rewritten in the form shown in eq. 7.18,

$$\begin{aligned}
 f = \{ & (M_0 = 1) \& (M_1 = 1), \\
 & (M_0 = 1) \& (M_0 = 1), \\
 & (M_1 = 1) \& (M_1 = 1), \\
 & (M_1 = 1) \& (M_0 = 1) \}
 \end{aligned} \tag{7.18}$$

and finally becomes $f = [m_0m_1, m_0m_0, m_1m_1, m_1m_0]$. Here $m_n = 1$ is the result of measurement with value n (example: when the measured for value

1 (M_1^0 is used), $m_1 = 1$ means that the outcome is 100% correct value 1 on this qubit. Note that in Figures the Measurement-Dependent Observable is denoted by M_n ; for instance M_n located on qubit indexed 0, $M_n = M_n^0$.

Example 7.5.0.2 Measurement Dependent Function I

Assume the following state encoding

$$\begin{aligned}
 |00\rangle &\rightarrow \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\
 |01\rangle &\rightarrow \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\
 |10\rangle &\rightarrow \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\
 |11\rangle &\rightarrow \frac{|01\rangle - |10\rangle}{\sqrt{2}}
 \end{aligned} \tag{7.19}$$

is used for function f from equation 7.17 to represent each state $|\psi\rangle$ of the system. The function of the entanglement operator σ is represented in Figure 7.24.

In the case of the $|01\rangle \rightarrow \frac{|00\rangle + |11\rangle}{\sqrt{2}}$, the output projecting the system through M_1^0 , will generate unique state with 100% of the correct output ($|01\rangle \rightarrow M_1^0 \left| \frac{|01\rangle + |10\rangle}{\sqrt{2}} \right.$). For the case $|10\rangle \rightarrow \frac{|00\rangle - |11\rangle}{\sqrt{2}}$, the M_1^0 will collapse the system to $|11\rangle$. Thus the original function 7.17, can be now extracted from a more general logic generator function 7.18 and reduced to $f = [m_1^0, m_0^0, m_1^1, m_0^1]$. $\square$

Because $M_n|\psi\rangle$ represents the state of the system after measurement taken into account if $m_n = 1$, these observables can represent either a function depending or not depending on the measurement. Allowing to describe boolean reversible

Figure 7.24: The circuit representing function f (eq. 7.19) and represented as in eq. 7.18. Observe that in this case the measurement is controlled by quantum signals that are copied using a set of CNOT gates and measured afterwards. Unlike in Figure 7.21 this setup allows more freedom in the sense that the addressing mechanism can now select a cell from the ROM with a probability rather than deterministically.

function, partially or completely specified such as:

$$f = \sum_n S_n \quad (7.20)$$

with $S_n = \prod_k \frac{M_k^j |n\rangle}{\sqrt{\langle n | M_k^* M_k | n \rangle}}$ is the expected value of the desired output for the input state $|n\rangle$ in the expected output LUT/ROM. The measurement operator M_k^j from equation 7.20 is applied to circuit output state $|n\rangle = F|n'\rangle$ at index n projecting the whole system on axis $|j\rangle$.

Theorem 7.5.5 Single Qubit Measurement-Dependent Functional Representation

Let f be a reversible function defined over $[0, 1]^{\otimes n}$ and mapping onto $[0, 1]^{\otimes n}$, then f can be defined as a function of m_n . $\square$

Proof. The operators M_n do not alter the expectation probabilities of the

system being build from unitary operators. As M_n requires entanglement, m_n represents the normalized real value in the circuit and the output is either deterministic or probabilistic. Such output corresponds to unique pairs of values (for a given entanglement circuit) build from the result of M_n and from the addressing output (Figure 7.24). This unique representation allows thus to define reversible functions. $\square$

Example 7.5.0.3 Measurement Dependent Function II

The function from equation 7.17 can now be reduced as follows:

$$f = \{m_0m_1, m_0m_0, m_1m_1, m_1m_0\} \rightarrow m_0 = M_1|b\rangle \quad (7.21)$$

where $M_1|b\rangle = m_1^b$ represents the initial state of the qubit b (leftmost element in each pair from eq. 7.21). In more details, the function f is now represented as a function of σ operator and of the input measurement operator M_1 and the output of the computation is accepted every time when measurement on qubit b yields the same value as it was set to on the inputs. $\square$

Note that so far no methodology was invented by physicists that would allow to force a quantum system to collapse to a particular state. Thus the measurement scheme described above can be realized only probabilistically.

For instance the function described by equation 7.21 cannot be determined using a single measurement. This is because the fact that the observation of

the results of M_0 and M_1 is probabilistic. It implies that the measurement depending representation takes another view: to correctly describe the function, use only the outputs that coincide with the expected values. If the input to the function equation 7.21 is the state $|00\rangle$, the measurement must continue until the result is M_0 on the given qubit. Once this is obtained, the state of the system is in a state that can be processed further and the observer has 100% certainty of the output state.

Example 7.5.0.4 Measurement Dependent Function III

Let $f = [0, 1, 0, 1, 0, 1, 1, 0]$ be the function to be designed as a reversible, measurement - dependent circuit (it happens to be a specification of Toffoli gate). First, let's rewrite the function in the measurement dependent manner:

$$f = \{m_0, m_1, m_0, m_1, m_0, m_1, m_1, m_0\} \quad (7.22)$$

Using this definition as the output control qubit register it can however be seen that this definition is not canonical as it allows many implementations with respect to this output specification. One of the possible realizations is shown in Figure 7.25.

Observe that using the setup from Figure 7.25 and using the definition of f from eq. 7.22 to represent the function and select the single qubit measurement, the whole output state (as shown in Figure 7.25) becomes a relation because the input states $|000\rangle$ and $|111\rangle$, and the input states $|001\rangle$ and $|110\rangle$ have the same outputs $|000\rangle$ and $|001\rangle$ respectively. However, the qubit $|c\rangle$ of

Figure 7.25: Example of implementation of the Toffoli gate using measurement-dependent reversible implementation for a single bit correct output. In this case, observe that the single measured qubit represents the Toffoli gate, but the overall output (three bits) is now a relation. The selection of the pre-stored values in the output LUT uses the selection method from Figure 7.22, but here there are three measurements in addresses of ROM and only one qubit is conditionally measured ($|c\rangle$). Observe that for clarity the output ROM contains the whole output states, but the real content of the ROM is only the bold value that is used to select the appropriate measurement operator. Also, the output signals (A) are connected to the qubits of the circuit with dotted lines representing the fact that the output is selected only conditionally (based on the result of the measurement).

Toffoli gates has the cost of $4 \times 2 \times 2$ primitives while here it is realized with 2×2 primitives and one 1×1 primitive. The methods developed in this section can thus decrease the cost of quantum component of the entire system at the price of designing a more complex classical logic related to the measurement.

□

Figure 7.26: Another possible realization of the circuit realizing the measurement-dependent Toffoli function. (a) the circuit uses partial input state and the output of the measurement to select the correct output state. In this case the complete output states are stored in classical memory and the output of the circuit is used only for retrieving the content from the ROM. (b) a MUX representing the selection of the output based on input and single qubit measurement. Remark, this is different from the previous approach where the measurement is used to decide which of the possible outputs is the correct one. Similarly to Figure 7.25 the output signals are conditionally (measurement-outcome dependent) assigned to output wires (A). (Figure 7.21)

7.5.1 Conclusions on the Measurement Dependent Quantum Logic

Synthesis

Observe that the synthesis method that uses the measurement dependent approach can use a single entanglement circuit to synthesize any output function for a single qubit. This can be easily seen as the output of each measurement is 50% of $|0\rangle$ and 50% of $|1\rangle$; the desired non reversible mapping can be specified using the measurement operators such as $f = \{m_0, m_0, m_0, m_1\}$.

Also, the presented approach was used here only to illustrate the principle. My measurement-dependent methodology can be extended to use different

measurement operators (different expected quantum states) as well as it can be extended for POV measurements. Therefore, the presented approach is more a protocol allowing for different methods to synthesize quantum circuits rather than a complete methodology to synthesize circuits for quantum computing. Moreover, as most of the quantum algorithms use at least some type of partial entanglement, the measurement-dependent approach can be useful in designing novel methods of minimization and synthesis.

The presented measurement dependent approach can be shown in an alternate form (Figure 7.26). In this case the shown realization behaves as a multiplexer for which the addressing is done partially based on the inputs and partially done as the result of the measurement. This leads to the fact that the measurement can be used in various ways in the QLS. Here, the presented methods showed the following possibilities of measurement dependent quantum circuits:

- - the result of the measurement selects the value that allows to decide which global output state to use
- - the result of the measurement is a part of the addressing mechanism when selecting stored output values
- - the addressing mechanism is using classical signals or quantum signals

Finally, note that as the output of MDQ circuit is probabilistic, once the measurement is selected the output might not be what is expected. Thus the result of the MDQC is obtained by either repeating the whole computation

from the beginning or by using quantum tomography; each state is represented many times and thus one can do many various operations on the same quantum state. This requirement for selection also introduces time delay in the QLS process. This delay could be used in the future to present another type of behaviors related to the speed of selecting the correct output state for the actuator control.

8 DISSERTATION CONCLUSION AND CONTRIBUTIONS

In this dissertation we presented three topics related to Quantum Logic Synthesis.

First, I presented an Evolutionary approach to QLS and showed that using simple circuit encoding and a Genetic Algorithm designed by me, I was able to design novel realizations of universal reversible quantum circuits, minimize some of the already known realizations of universal functions and to describe the relation between the correctness of the synthesis process and the cost of the realized quantum circuits. This algorithm was also later used in the synthesis of quantum finite state machines as a proof of concepts described in this dissertation.

Second, from the results of the evolutionary synthesis I was able to extract heuristic rules for simple but efficient search algorithm, the EX. This algorithm allowed me to discover a family of inexpensive universal quantum gates having all the same cost. This family of quantum gates is based on the Peres gate and it is the least expensive known quantum universal gate family.

Finally, using the GA and observations about inductive bias gained during the usage of the EX algorithm, I formulated an inductive learning method based on QLS. In particular, I describe the synthesis and the learning of quantum robotic behaviors (called Quantum Braitenberg Vehicles) and I also show that

this approach allows to generate novel behaviors, otherwise not observable in classical or probabilistic learning. Moreover, in Appendix C I present a robotic framework that I built to illustrate my QLS and machine learning approaches. The evolutionary algorithm performance showed that in order to scale the presented approach over the tested 4-qubits, the algorithm requires both usage of larger macros (such as N-Toffoli) in order to be performant with respect to other synthesizers. Similarly the EX algorithm can be upgraded to perform local searches for larger quantum circuits.

The algorithms that have been presented demonstrated that the machine learning proposed in this dissertation is based on the circuit representation. Moreover the measurement-based synthesis uses some aspects that are strongly influenced by technology. A realization of an algorithm in optical, Cavite QED or NMR are three quite different approaches requiring a different measurement setting and apparatus setting. Thus minimizing a computer based on measurement with respect to measurement does not only imply a change in programming the quantum computer but also in changing the controlling classical computer. Thus the measurement based approach might in the future require a particular approach dealing with both components.

From the global point of view, the work done in this dissertation is not only a study in QLS but also is an computational and experimental approach to the exploration of novel applications for quantum computing. In particular, the notion of the cost of the quantum circuits was demonstrated to have multiple

meanings in quantum technology: the cost of quantum computation, the cost of quantum initialization, the cost of quantum measurement (result capture) or the cost of the classical computation that controls the quantum computer. Thus depending on the technology the logic synthesis can be formulated to minimize any of these components of quantum circuits.

This novel approach to Logic Synthesis entails that the application of quantum circuits can be done not only with respect to a faster computation (Grover or Shor algorithm) but also with respect to a particular computational paradigm. As was shown it is for instance possible to synthesize logic functions by minimizing the measurement operations. Another example can be a quantum computer of very large size such as a human brain where the computation is redundant but the result is only statistically confirmed - minimization with error in a fixed architecture.

To conclude the presented work successfully demonstrates that my heuristic - algorithmic method allowed to discover the principle of cost based minimization, cost based QLS, structure based QLS and measurement dependent machine learning.

REFERENCES

- [AP72] S. Amoroso and Y. N. Patt. Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *J. Comp. Sys. Sci*, 448(6), 1972.
- [AS07] A. Atici and R.A. Servedio. Quantum algorithms for learning and testing juntas. *Quantum Information Processing*, 6:323, 2007.
- [Ati06] A. Atici. *Advances in quantum computational learning theory, Ph.D. dissertation*. Department of Mathematics, Columbia University, 2006.
- [AW02] A. Ambainis and J. Watrous. Two-way finite automata with quantum and classical states. *Theoretical Computer Science*, 287(1):299–311, 2002.
- [Bak87] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *In Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21, 1987.
- [BBC⁺95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and Weinfurter H. Elementary gates for quantum computation. *The American Physical Society*, 5:3457–3467, 1995.

-
- [Ben82] P. Benioff. Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.
- [Ben98a] P. Benioff. Quantum robots and environments. *Physical review. A*, 55(2):893–904, 1998.
- [Ben98b] P. Benioff. Some foundational aspects of quantum computers and quantum robots. *Superlattices and microstructures*, 23(3-4):407–417, 1998.
- [Ben02] P. Benioff. Space searches with a quantum robot. *AMS Contemporary math series*, 305, 2002.
- [BJ95] N. H. Bshouty and J. C. Jackson. Learning DNF over the uniform distribution using a quantum example oracle. In *COLT '95: Proceedings of the eighth annual conference on Computational learning theory*, pages 118–127, New York, NY, USA, 1995. ACM Press.
- [Bra86] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1986.
- [Bre02] C. Breazeal. *Designing Sociable Robots*,. MIT Press, 2002.
- [BZ00] A. Blais and A. M. Zagoskin. Operation of universal gates in a solid state quantum computer based on clean josephson junctions between d-wave superconductors. *Phys. Rev. A*, 61, 2000.

-
- [CFH97] D. G. Cory, A. F. Fahmy, and T. F. Havel. Ensemble quantum computing by NMR spectroscopy. *Proc. Natl. Acad. Sci. USA*, 94:1634–1639, 1997.
- [CZ95] J.I. Cirac and P. Zoller. Quantum computation with cold trapped ions. *Physical Review letters*, 74(20):4091, 1995.
- [Dam99] A. R. Damasio. *The Feeling of What Happens: Body and Emotion in the Making of Consciousness*. Harcourt 1999, 1999.
- [Dam04] A. R. Damasio. *Looking for Spinoza*. Picador, 2004.
- [Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Ser. A*, A400:97–117, 1985.
- [Dir84] P. A. M. Dirac. *The principles of quantum mechanics*. Clarendon, Oxford, 1984.
- [DiV95] P. DiVincenzo. Two-bit gate for quantum computation. *Physical Review A*, (50):1015, 1995.
- [DJ92] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proc. Royal Society of London*, A400:73–90, 1992.
- [DKK03] L.M. Duan, A. Kuzmich, and H.J. Kimble. Cavity QED and quantum-information processing with 'hot' trapped atoms. *Physical Review A*, 67, 2003.

-
- [Ekm93] P. Ekman. Facial expression and emotion. *American Psychologist*, 48(4):384–392, 1993.
- [Ekm03] P. Ekman. Darwin, deception, and facial expression. *Annals of the New York Academy of Sciences*, 1000:205–221, 2003.
- [EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.
- [EvKK95] A. E. Eiben, C. H. M. van Kemenade, and J. N. Kok. Orgy in the computer: multi-parent reproduction in genetic algorithms. In *94*, page 10. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1995.
- [Fey85] R. P. Feynmann. Quantum mechanical computers. 11:11, 1985.
- [FH65] R. P. Feynman and A. R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill, 1965.
- [Fil98] C. Files. Decomposition of multiple-valued functions for machine learning, vlsi layout generators, and FPGAs, 1998.
- [FP98] C. M. Files and M. A. Perkowski. An error reducing approach to machine learning using multi-valued functional decomposition. In *Proceedings of the International Symposium on Multiple-Valued Logic 1998*, pages 167–172, 1998.

-
- [GKD89] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3:493–530, 1989.
- [Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, MA, 1989.
- [Gra81] A. Graham. *Kronecker Products and Matrix Calculus With Applications*. Ellis Horwood Limited, Chichester, U.K., 1981.
- [Gro96] L.K. Grover. A fast quantum mechanical algorithm for database search, 1996.
- [Gru99] J. Gruska. *Quantum computing*. Osborne/McGraw-Hill, U.S., 1999.
- [Gry00] S. Grygiel. *Decomposition of Relations as a new approach to constructive Induction in Machine Learning and Data Mining*. PhD dissertation, Portland State University, 2000.
- [Hol75] J.H. Holland. *Adaptation in Natural and Artificial systems*. The University of Michigan Press, Ann Arbor, 1975.
- [HPLK01] K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1422–1429, COEX, World

-
- Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.
- [HSY⁺04] W.N.N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Provably optimal reversible quantum logic synthesis via symbolic reachability analysis. In *Proceedings of DAC*, 2004.
- [HSY⁺06] W.N.N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and systems*, 25(9):1652–1663, 2006.
- [Kak95] S. C. Kak. On quantum neural computing. *Information Sciences*, 83(3&4):143–160, 1995.
- [Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [Koz94] J.R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.
- [KP06] F. S. Khan and M. A. Perkowski. Synthesis of multi-qudit hybrid and d-valued quantum logic circuits by decomposition. *Theoretical Computer Science*, 3(367):336–346, 2006.
- [KPK02] A. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible logic

- synthesis by gate composition. In *Proceedings of IWLS*, pages 261–266, 2002.
- [KW97] A. Kondacs and J. Watrous. On the power of quantum finite state automata. In *IEEE Symposium on Foundations of Computer Science*, pages 66–75, 1997.
- [Lan61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.
- [Lan02] C. Lang. *Bi-Decomposition of functions using Multi-Valued Logic*,. PhD dissertation, University of Freiberg, Germany, 2002.
- [LBA⁺08] B. P. Lanyon, M. Barbieri, M. P. Almeida, T. Jennewein, T. C. Ralph, K. J. Resch, G. J. Pryde, J. L. O’Brien, A. Gilchrist, and A. G. White. Quantum computing using shortcuts through higher dimensions. *ArXiv e-prints*, 804, April 2008.
- [LLK⁺06] S. Lee, S-J. Lee, T. Kim, J-S. Lee, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple Valued Logic and Soft Computing*, 12(5/6):561–574, 2006.
- [LP02] M. Lukac and M. Perkowski. Evolving quantum circuit using genetic algorithm. In *Proceedings of the 2002 NASA/DoD Conference on Evolvable hardware*, pages 177–185, 2002.

-
- [LP05a] M. Lukac and M. Perkowski. Combining evolutionary and exhaustive search to find the least expensive quantum circuits. In *Proceedings of ULSI symposium*, 2005.
- [LP05b] M. Lukac and M. Perkowski. Using exhaustive search for the discovery of a new family of optimum universal permutative binary quantum gates. In *Proceedings of International Workshop on Logic & Synthesis, Poster Session*, 2005.
- [LP07] M. Lukac and M. Perkowski. Quantum mechanical model of emotional robot behaviors. In *Proceedings of the International Symposium on Multiple-Valued Logic*, 2007.
- [LPG⁺03] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B.-G. Kim, and Y.-D. Kim. Evolutionary approach to quantum reversible circuit synthesis. *Artif. Intell. Review.*, 20(3-4):361–417, 2003.
- [LPG⁺04] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim. Evolutionary approach to quantum and reversible circuits synthesis. In *Artificial Intelligence in Logic Design*, pages 201 – 257. Kluwer Academic Publisher, 2004.
- [LPMP02] M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski. Automated synthesis of generalized reversible cascades using genetic

-
- algorithms. In *Proceedings of Fifth Intern. Workshop on Boolean Problems*, pages 33–45, 2002.
- [MC00] C. Moore and J.P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237:275–306, 2000.
- [MD03] D. M. Miller and G.W. Dueck. Spectral techniques for reversible logic synthesis. In *Proc. RM*, pages 56–62, 2003.
- [MD04] D. Maslov and W. Dueck. Improved quantum cost for n-bit Toffoli gates, 2004.
- [MDM05] D. Maslov, G. W. Dueck, , and D. M. Miller. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Transactions on VLSI*, 13(6):765–769, 2005.
- [Mil02] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Proc. Midwest Symposium on Circuits and Systems, on CD-ROM*, August 2002.
- [MM04] D. C. Marinescu and G. M. Marinescu. *Approaching quantum computing*. Prentice Hall, 2004.
- [MM06] D. M. Miller and A. T. Mitchell. QMDD: A decision diagram structure for reversible and quantum circuits. In *Proc. 2006 Int. Symposium on Multiple-Valued Logic*, 2006.

-
- [MMD06] D. M. Miller, D. Maslov, and G. W. Dueck. Synthesis of quantum multiple-valued circuits. *Journal of Multiple-Valued Logic and Soft Computing*, 12(5-6):431–450, 2006.
- [MMKW96] C. Monroe, D. M. Meekhof, B. E. King, and D. J. Wineland. A schrödinger cat superposition state of an atom. *Science*, 272, 1996.
- [MML⁺98] C. Monroe, B. E. Meekhof, D. M. ind King, D. Leibriefd, W. M. Itano, and D. J. Wineland. Manipulating the motion of a single trapped atom, 1998.
- [Moo65] G.E. Moore. Cramming more components onto integrated circuits. In *Electronics, April 19*, 1965.
- [MP02] A. Mischenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *Proceedings of IWLS*, pages 197–202, 2002.
- [MS00] A. Muthukrishnan and C. R. Stroud. Multivalued logic gates for quantum computation. *Phys. Rev. A*, 62(5):052309, Oct 2000.
- [MT78] S. Martello and P. Toth. Algorithm for the solution of the 0-1 single knapsack problem. *Computing*, (21):81–86, 1978.
- [NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

-
- [NM96] A. Narayanan and M. Moore. Quantum inspired genetic algorithms. In *International Conference on Evolutionary Computation*, pages 61–66, 1996.
- [OS99] T. Ogata and S. Sugano. Emotional communication between humans and the autonomous robot which has the emotion model. In *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, volume 4, pages 3177 – 3182, 1999.
- [OS00] T. Ogata and S. Sugano. Emotional communication robot: Wamoeba-2r emotion model and evaluation experiments, 2000.
- [PARK⁺01] M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, M. Md. Azad Khan, A. Coppola, S. Yanushkevich, V. Shmerko, and L. Jozwiak. A general decomposition for reversible logic. In *Proc. RM'2001*, August 2001.
- [Per00] A. Peres. Reversible logic and quantum computers. *Physical review*, (32):3266–3276, 2000.
- [PW02] J. Pachos and H. Walther. Quantum computation with trapped ions in an optical cavity. *Physical Review Letters*, 89(18), 2002.
- [Rai96] G. Raidl. *Evolutionary Algorithms*. University of Technology, Wien, 1996.

-
- [RFW⁺07] A. Raghuvanshi, Y. Fan, M. Woyke, A. Kumar, and M. Perkowski. Quantum robots for teenagers. In *Proceedings of the International Symposium on Multiple-Valued Logic 2007*, 2007.
- [Rub00] B. Rubinstein. *Evolving Quantum Circuits using Genetic Programming*, pages 325–334. Stanford University, 2000.
- [SBM05a] V. V. Shende, S. S. Bullock, and I. L. Markov. A practical top-down approach to quantum circuit synthesis. In *Proceedings of Asia Pacific DAC*, 2005.
- [SBM05b] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum logic circuits. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 272–275, New York, NY, USA, 2005. ACM Press.
- [SC02] M. Sasaki and A. Carlini. Quantum learning and universal quantum matching machine. *Phys. Rev. A*, 66(2):022303, Aug 2002.
- [SCJ01] M. Sasaki, A. Carlini, and R. Jozsa. Quantum template matching. *Phys. Rev. A*, 64(2):022317, Jul 2001.
- [SD96] J. Smolin and D. P. DiVincenzo. Five two-qubit gates are sufficient to implement the quantum fredkin gate. *Physical Review A*, 53(4):2855–2856, 1996.
- [Ser01] R.A. Servedio. Separating quantum and classical learning. *Lecture Notes in Computer Science*, 2076:1065–1080, 2001.

-
- [SG01] R.A. Servedio and S.J. Gortler. Quantum versus classical learnability. In *Computational Complexity, 16th Annual IEEE Conference on*, pages 138–148, 2001.
- [Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science (Shafi Goldwasser, ed.)*, pages 124–134. IEEE Computer Society Press, 1994.
- [TM90] T. Toffoli and N. Margolus. Invertible cellular automata: a review. *Physica, D* 45:229–253, 1990.
- [Tof77] T. Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer Systems Science*, 15:213–231, 1977.
- [Val84] L.G. Valiant. A theory of learnable. In *Proceedings of the sixteenth annual ACM Symposium on Theory of Computing*, pages 436–445. New York, NY, USA, 1984.
- [VM97] D. Ventura and T. Martinez. An artificial neuron with quantum mechanical properties. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, 1997.
- [VM98] D. Ventura and T. Martinez. Quantum associative memory with

- exponential capacity. In *Proceedings of the International Joint Conference on Neural Networks*, pages 509–513, 1998.
- [Wat95a] J. Watrous. On one-dimensional quantum cellular automata. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 528–537, 1995.
- [Wat95b] J. Watrous. On one-dimensional quantum cellular automata. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 528–532, 1995.
- [Wat97] J. Watrous. On the power of 2-way quantum finite state automata. Technical Report CS-TR-1997-1350, 1997.
- [WG98] C. Williams and A. Gray. Automated design of quantum circuits. In *in Proceedings of QQC 1998*, pages 113–125, 1998.
- [Yen05] Ö. Yenyay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and computational Applications*, 10(1):45–56, 2005.
- [YHSP03] G. Yang, W.N.N. Hung, X. Song, and M. Perkowski. Majority-based reversible logic gate. In *Proceedings of 6th International Symposium on Representations and Methodology of Future Computing Technology, March 10-11, Trier, Germany*, pages 191–200, 2003.

-
- [You95] S. Youssef. Quantum mechanics as an exotic probability theory. In *Workshop on Maximum Entropy and Bayesian Methods, St. John's College, Santa Fe, New Mexico*, August 1995.
- [YSHPO5] G. Yang, X. Song, W. N. N. Hung, and M. Perkowski. Fast synthesis of exact minimal reversible circuits using group theory. In *Proceedings of Asia Pacific DAC*, pages 1002 – 1005, 2005.
- [YSPH05] G. Yang, X. Song, M. Perkowski, and W. N. N. Hung. The power of large pulse-optimized quantum libraries: Every 3-qubit reversible function can be realized with at most four levels. In *Proceedings of Proc. of IWLS*, 2005.
- [YSPW05] G. Yang, X. Song, M. Perkowski, and J. Wu. Realizing ternary quantum switching networks without ancilla bits. *Journal of Physics A, Mathematical and General*, pages 9689–9697, 2005.
- [ZBBD97] B. Zupan, M. Bohanec, I. Bratko, and J. Demšar. Machine learning by function decomposition. In *Proc. 14th International Conference on Machine Learning*, pages 421–429. Morgan Kaufmann, 1997.
- [ZLSD02] J. Zhang, Z. Lu, L. Shan, and Z. Deng. Realization of generalized quantum searching using nuclear magnetic resonance. *Physical Review A*, 65:034301.1–034301.4, March 2002.

-
- [Zup97] B. Zupan. *Machine learning based on function decomposition*,. PhD dissertation, University of Ljubljana, 1997.

APPENDICES

A GLOSSARY

Quantum Register/Array is name for a set of qubits that can be used as a basis for building quantum circuit. In particular it refers to a set of qubits without any additional mechanism such as Initialization or Unitary transforms.

Quantum Circuit is the name given to a Quantum Register/Array with additional mechanisms. In particular every quantum circuit must have (beside a quantum register) a controllable Initialization protocol, a controllable Computation protocol and a controllable Measurement protocol. This means that a quantum circuit allows to do controlled computation on a quantum array.

Quantum Logic Block refers to a set of quantum gates that can be concatenated using the collapsing method described in section 3.6.4.

Quantum Logic Segment refers to a block of quantum logic gates spanning the whole width of the given quantum circuit.

Quantum Evolution represents the process of quantum computation. In other words quantum evolution refers here to the controlled

Quantum Circuit Cost refers to the sum of cost of all gates that are used to build this quantum circuit. The resulting cost depends thus on the number of the quantum gates as well as on the cost of each gate.

Quantum Logic Primitive is the name for the smallest units of logic reasoning used to build quantum circuits. Logic primitive can be a single or multiple qubit quantum gate as well as a macro constructed from smaller quantum gates.

Quantum Logic Gate represents a logic operations used to build larger quantum gates and quantum circuits. In general quantum gates can be quantum primitives but this is not required.

Chromosome is the container of the information within each individual. In general it can be a binary string, a string of integers or of floats.

Individual is the abstract representation of potential solution in a GA. An individual contains a chromosome.

Genotype and Phenotype The genotype is the encoding of the information

within an individual. The phenotype is the expression of the genotype of an individual. This is meaningful mainly when the expression (the phenotype) is different than the genotype.

B MODELS OF QUANTUM COMPUTING

The paradigms of quantum circuits from Section 2.5 have been studied and a natural extension to FSM was shown which was created for use in evolutionary and exhaustive search algorithms. In this Appendix we describe few computational models such as Finite State Machines (FSM); Cellular Automata (CA), Turing Machines (TM) and their quantum counterparts. Most of the work in this area is still on the theoretical level but the proofs of concepts allow to speculate that such models will be useful for quantum logical devices that will appear in close future.

B.0.2 Quantum Cellular Automata

Definition B.0.6 Classical Cellular Automata

The classical Cellular Automaton (CA) (of dimensionality d) M is defined:

- a finite set of states Q ,
- a scheme of finite neighborhood $N = \{N_1, \dots, N_{|N|}\}$, $N \in \mathbb{Z}^d$ and such that $N_1 < N_2 < \dots < N_{|N|}$,
- a local transition function $\gamma : Q^{|N|} \rightarrow Q$

□

The scheme of the neighborhood N , contains the set of all possible neighborhood schemes on a neighborhood of size N and $N_i \in \mathbb{Z}^1$. For example a neighborhood can take the form such that: $N_1 = \{0, 1\}$ or $N_2 = \{-1, 0, 1\}$ (Figure B.1c).

A CA is generally described by the configuration and the local state transition function is applied at each step to its state.

Definition B.0.7 Configuration

A configuration $C \in \{Q\}^{\otimes \mathbb{Z}^d}$ is the total state of the automaton given by the

¹ $\mathbb{Z}$ being the set of integers

Figure B.1: Three types of neighborhood schemes. The two leftmost schemes represent the left neighbor and itself and the right neighbor and itself. The rightmost neighborhood is based on three cell states: the left neighbor, the right neighbor and the cell itself, respectively. The numbering of the cells is made with respect to the current cell; to the right positive and to the left negative.

(local) states of the individual cells $c_i(k)$. Thus the whole configuration can be thought of as a vector in d dimensional space, and each step is a rotation. Examples of configurations of CA can be seen in Table B.2 or B.3; for each step the configuration of the CA is shown in the appropriate row. $\square$

Example B.0.2.1 CA Evolution

Assume a CA K , with the local transition function represented in Table B.1.

Table B.1: Function $c = a \oplus b$

a	b	c
00		0
01		1
10		1
11		0

A simple neighborhood scheme $n_i = \{0, 1\}$ and the set $Q = \{0, 1\}$ of cell states. Finally let's set up the CA to an initial state (set all but one cells to 0 and one cell to 1), (Table B.2).

At each step, the CA moves from one of its possible configurations to another. For instance taking the initial configuration at time $t = 0$, cell at index 5 is set to 1, and the transition to the next step given n_i means: take cells at indexes 4 and 5, and apply the function from Table B.1. Thus $(c_4, c_5) = (0, 1) \rightarrow f(0, 1) = 1$. Thus cell at index 4 c_4 will change its value from 0 to 1. Each time a cell is at index 0 in the rule, the output of the local transition function is applied to it. Also observe that because the transition function does not change the state when $(c_i, c_{i+1}) = (0, 0)$, thus the analysis of the

Table B.2: A CA with $N = 0, 1$

	0	1	2	3	4	5	6	7	8	9	10	11
t = 0	0	0	0	0	0	1	0	0	0	0	0	0
t = 1	0	0	0	0	1	1	0	0	0	0	0	0
t = 2	0	0	0	1	0	1	0	0	0	0	0	0
t = 3	0	0	1	1	1	1	0	0	0	0	0	0

state transition is limited to cells in which the neighbors or the central cell is 1.

With a different neighborhood scheme $n_i = \{-1, 0\}$ the same CA will behave as shown in Table B.3.

Table B.3: Function $a \oplus b$ with $N = \{-1, 0\}$

	0	1	2	3	4	5	6	7	8	9	10	11
t = 0	0	0	0	0	0	1	0	0	0	0	0	0
t = 1	0	0	0	0	0	1	1	0	0	0	0	0
t = 2	0	0	0	0	0	1	0	1	0	0	0	0
t = 3	0	0	0	0	0	1	1	1	1	0	0	0

□

When analyzing CA from the complexity point of view, it is useful to represent the evolution of the (probabilistic) CA in terms of the probability of going from one configuration to another:

$$p(c_1, c_2) = \prod_{m \in \mathbb{N}} \gamma(c_1(m + n_1), \dots, c_1(m + n_r), c_2(m)) \quad (\text{B.1})$$

where $c_n(m + n_k)$ is the state of the cell at the index $m + n_k$ in the configuration C_n . The eq. B.1 quantifies the configuration transition by representing the probability of changing the current configuration c_1 to c_2 with probability given by the product of probabilities of each cell in $c_1(m)$ changing to $c_2(m)$. Such description with respect to the whole CA, allows to analyze the convergence of CA's behavior. For deterministic reversible CA's this probability is either 1 or 0.

The transition from a configuration to the next configuration is executed via the local transition function.

Definition B.0.8 Local Transition Function

The local transition function of a CA, can be specified with respect to a neighborhood configuration as a product:

$$\Gamma(C) = \Gamma(\otimes_{i \in Z} c_i) \quad (\text{B.2})$$

with $\otimes_{i \in Z} c_i$ represents the neighborhood of a given cell. For the CA from Example B.0.2.1 the global outcome is obtained by applying the function from Table B.1 to a configuration. $\square$

Example B.0.2.2 Local Transition Function

For example let $\gamma : Q^3 \rightarrow Q$ defined by:

$$\begin{aligned} \gamma(0, 0, 0) &= 0 & \gamma(1, 0, 0) &= 1 \\ \gamma(0, 0, 1) &= 0 & \gamma(1, 0, 1) &= 1 \\ \gamma(0, 1, 0) &= 1 & \gamma(1, 1, 0) &= 0 \\ \gamma(0, 1, 1) &= 0 & \gamma(1, 1, 1) &= 1 \end{aligned} \quad (\text{B.3})$$

This local function corresponds to the logic function $f = a \oplus bc'$. $\square$

From a global point of view, the local function γ can be extended to the overall configuration of the CA.

Definition B.0.9 The Global Transition Function

The global transition function of a CA, is obtained as the function describing the change from a configuration c_j to c_k . Thus the global transition function is represented with respect to time:

$$(s, t) \rightarrow [\Gamma_Z^t(C)]_s \quad (\text{B.4})$$

with s the initial configuration of the CA, t the number of steps from the initial state represented by the number of times the transform is applied over the whole CA, and C being the set of all configurations. This process is illustrated in Table B.2 and B.3. $\square$

Before turning to quantum cellular automata let us define an Reversible Cellular Automaton (RCA) [Tof77, TM90].

Definition B.0.10 RCA

A *Reversible* CA is such a CA that allows unique mapping $F(S) = Y$ and $F^{-1}(Y) = S$, with S and Y being respectively the set of binary vectors defined on dimension d. The input set of vectors S and the output set O are from the set of all configuration C of the RCA. Thus for an RCA to be reversible defined by the transition function eq. B.5 it must also hold that if two configurations C_k and C_{k+1} satisfy eq. B.5

$$\Gamma(C_k) = \Gamma(\otimes_{i \in Z} C_i) = C_{k+1} \quad (\text{B.5})$$

then

$$\Gamma^{-1}(C_{k+1}) = C_k \quad (\text{B.6})$$

□

It is possible to distinguish the reversible CA between those that are locally reversible but are not the reversible CA and those CA that are fully reversible. A *locally reversible* RCA is such that $f(s_a, \dots, s_b)$ is reversible, however the local reversibility does not necessarily guarantee the global reversibility (see Example B.0.2.3 below). A *globally reversible* RCA is similarly defined such that $F(S) = Y$ (mapping from one configuration to another of the whole automaton) is reversible; i.e. the configuration Y has only one precedent.

The trivial case of an RCA is when the neighborhood for each cell includes only itself. In such a case, the reversibility of the whole CA is given solely by the reversibility of the local function $q'_1 q'_2 q'_3 = h(q_1, q_2, q_3)$.

Example B.0.2.3 Locally RCA

Assume the reversible function from Table B.4. Let K be a CA in configuration c such that $c = \{0, 0, 1, 0, 1, 0\}$ and the function h is the local transition function. The application of γ to the configuration c can be seen in Figure B.2. As can be observed the cell at index 3 is changed twice in this configuration and thus does not satisfy the definition. This can be seen as follows: let a CA of length 6 be in a configuration C_i (Figure B.2). Let $c_{010} = \{s_1, s_2, s_3\}$ and $c_{101} = \{s_2, s_3, s_4\}$ be the configurations for the cells at indices 2 and 3 in

Table B.4: 3×3 reversible function

$ q_1q_2q_3\rangle$	h	$ q_1q_2q_3\rangle$	h
000	000	100	011
001	001	101	110
010	010	110	101
011	100	111	111

Figure B.2. Using the function from Table B.4 one can see that the cell at index 3, has different next state for $h(c_{010})_1 = 0$ and for $h(c_{101})_0 = 1$. Thus if one would update all cells in parallel, conflicts will appear (as shown in cell 3 Figure B.2).

Figure B.2: An example of a non reversible cellular automaton that uses a reversible local function from Table B.4. a) - Observe the cell at position 3. From left to right, the cell is first changed to 0 then to 1. b) - the cells where the global reversibility is violated are shown by arrows. c) - a lattice view: dashed lines represent 0/white, full lines represent 1/black. Each time when the next state lines do not agree the global reversibility of the RCA is lost.

□

Because of the above illustrated structural constraints the globally reversible CAs are, in general, rare and simple [Gru99, Tof77, TM90, AP72].

A Quantum Cellular Automata (QCA) were developed by Watrous [Wat95a] as the natural extensions of classical Reversible Cellular Automata (CA). There are two main types of QCA: the one dimensional QCA (1QCA) and the Partitioned QCA (PQCA) [Wat95b]. First let us define the one dimensional 1QCA (qQCA) as the basic model of QCA's called the quiescent QCA.

Definition B.0.11 Quiescent 1QCA

The *quiescent* 1QCA is defined by a four tuple $A = (Q, \theta, N, \delta)$, where

- Q is the set of states, including the quiescent state θ defined as:

$$\delta : Q^{r+1} \rightarrow C^{r+1} \quad (\text{B.7})$$

and satisfying the local probability condition $\sum_q |\delta(q_1, \dots, q_r, q)|^2 = 1$, the quiescent state stability condition:

$$\delta(q_1, \dots, q_r, q) = \begin{cases} 1 & \text{if } q = \theta \\ 0 & \text{otherwise.} \end{cases} \quad (\text{B.8})$$

The quiescent state is a state that the automaton cannot escape from; i.e. an attractor such that if all cells are in state θ the automaton will remain blocked with all cells being in the state θ .

- N (as in classical case of CA) is the neighborhood scheme such that $N = \{n_1, \dots, n_r\} \subset \mathbb{Z}$
- $\delta : Q^{r+1} \rightarrow \mathbb{C}_{[0,1]}$ (complex Hilbert space) is the state transition function with r being the size of the neighborhood for a given cell in the 1QCA.

□

The automata of the 1QCA type can be classified based on the neighborhood N into three types:

- a *trivial neighborhood automata*, $N = \{0\}$, the state of the cell is based solely on its previous state (thus the automaton implements N single input/output functions on N cells),
- a *simple neighborhood* is defined on an interval $n_k = n_1 + i - 1$ for $i \in [1, r]$. For example $N = \{0, 1\}$ or $N = \{0, 1, 2\}$,
- and *symmetric neighborhood* such that $n_r = -n_1$, for example $N = \{-1, 0, 1\}$.

Similarly to classical CA, in any QCA the configuration represents the evolution of the initial state of the automaton with respect to the set of quiescent states. The quiescent state is used to study any CA or QCA when studying its computational properties. However, automata without a quiescent state (performing infinite operations) are also considered.

Thus a configuration C of A is defined as

$$c : \mathbb{Z} \rightarrow Q; \text{ such that } c(i) \neq \theta \text{ for finite } i \quad (\text{B.9})$$

and indicates that it concerns automata only with the finite length of execution time.² To describe the QCA as a quantum system, let $C(A)$ be the set of all possible configurations, a single configuration can be then represented as:

$$|\psi\rangle = \sum_{c \in C(A)} \alpha_c |c\rangle \quad (\text{B.10})$$

This means that the QCA can be described as a single wave equation (as a whole quantum system). The unitary evolution of such defined system is then made via an operator U_A acting on A :

$$|\psi\rangle = U_A |\tau\rangle = \sum_{c \in C(A)} \beta_c |c\rangle \quad (\text{B.11})$$

with $\beta_c = \sum_{c' \in C(A)_K} p_{c'} p(c', c)$.

Example B.0.2.4 QCA

Let $A = \{Q, \theta, N, \delta\}$ be a QCA of finite size $r = 3$ with neighborhood scheme $N = \{0, 1\}$ and the local quantum transition function defined by

$$\delta : U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 \end{pmatrix} \quad (\text{B.12})$$

with the current cell qubit being at the index 0 in the neighborhood scheme N . This means that for a two-qubit quantum state the qubit we are describing is on the left side of the ket. If one looks now for the next configuration of every

² There also exists QCA as well as CA with infinite cycle and can be referred to as chaotic CA/QCA.

configuration, then for instance for the QCA configuration $c_3 = |010\rangle$ the next configuration is described by the application of the 2×2 unitary operator from eq. B.12 to each neighbor pair of cells (identified but indexes of the qubits within the QCA):

$$\begin{aligned}
 |010\rangle &\rightarrow U|01\rangle_{21} + U|10\rangle_{10} + U|00\rangle_{02} \\
 &\rightarrow \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)_{21} \\
 &\quad + \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)_{10} \\
 &\quad + \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)_{20}
 \end{aligned} \tag{B.13}$$

The resulting state transitions cannot be represented as a three-qubit quantum CA because the parallel application of U to each of the pair of qubits will result in different global states. For instance

In order to describe a larger QCA in such a manner, a larger unitary matrix (assuming a larger neighborhood can be constructed. For instance, a three-qubit QCA state transition matrix is shown in eq. B.14. Observe that the problem of constructing QCA for larger dimensions entails the same problems as in the case of the RCA. Moreover, the notion of the neighborhood becomes unclear once the QCA is fully described by a Unitary matrix as a global state transition function.

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{B.14}$$

This is because, a QCA evolving using a Unitary matrix for computation does not need the neighborhood to function properly. Unlike in the classical CA where the neighborhood was crucial during the specification of the local

transition function, it can be seen that in the case of globally reversible CA and QCA this local specification can be omitted when the global transition function is specified using a single Unitary operator describing all the cells of the automaton at once. $\square$

Finally, the amplitude of state transition is given by the probability of transition between two basis states (configurations) c_1 and c_2 such that:

$$p(c_1, c_2) = \prod_{m \in \mathbb{N}} \delta(c_1(m + n_1), \dots, c_1(m + n_r), c_2(m)) \quad (\text{B.15})$$

representing the product of amplitudes of each cell of the automaton in configuration c_1 to pass from state $|\psi_{c_1}\rangle$ to state $|\psi_{c_2}\rangle$.

For instance, let us assume a QCA with size $s = 2$ represented by the global state $|\phi_0\phi_1\rangle$ and a neighborhood scheme $n = \{0, 1\}$. For two configurations $c_1 = \{|01\rangle\}$ and $c_2 = \{|10\rangle\}$ we can write the probability of transition (from eq. B.15):

$$p(|01\rangle, |10\rangle) = \delta(c_1(0), c_1(1), c_2(0)) \times \delta(c_1(1), c_1(0), c_2(1)) \quad (\text{B.16})$$

From the above it can be seen that a trivial 1QCA can be represented by a unique unitary operator ($V = \otimes^n V_i$, $i \in 0, \dots, n-1$) specifying transitions of all cells for any possible configuration, or in another words from one configuration $|c_1\rangle$ to $|c_2\rangle$ (for each q_{i_n}). This is a natural consequence of the fact that in such a case for V to be unitary the following must hold:

$$\sum_q \delta^*(q_1, q) \delta(q_2, q) = \begin{cases} 1 & \text{if } q_1 = q_2 \\ 0 & \text{otherwise.} \end{cases} \quad (\text{B.17})$$

In such case the global function as well as the local function of the QCA are both reversible. Thus, for example having a quantum cellular automaton T of size 3 (three cells), with $n_T = \{0\}$ being the neighborhood scheme, and a local quantum transition function $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, the CA behavior can be fully specified by a unitary operator such as:

$$V = X^{\otimes 3} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (\text{B.18})$$

Such an operator can also be represented as a quantum logic circuit (Figure B.3).

Figure B.3: Circuit representation for the Unitary operator $X^{\otimes 3}$.

The PQCA is very similar to standard QCA but instead of having cells with a single state, it represents the state of each cell with a vector of three values: left, middle, right. Figure B.4 shows a schematic representation.

Definition B.0.12 PQCA

The PQCA is a QCA $M = \{Q, \theta, N, \delta\}$ with the condition that the set of states of M is a Cartesian product of sets of states of each cell in the automaton $Q = Q_1 \times \cdots \times Q_r$. Also, the state transition function δ is composed from two sub-functions:

$$\begin{aligned} \delta_c : Q^r &\rightarrow Q \\ \delta_q : Q &\rightarrow C_{\{0,1\}} \end{aligned} \quad (\text{B.19})$$

To see how the PQCA works let's have a look at an example. In particular, let's have a look on the state transition function from eq. B.19 using the unitary matrix from eq. B.18. The PQCA has by default a fixed neighborhood of three $N = \{-1, 0, 1\}$. Thus for a state of three cells from Figure B.4 the

Figure B.4: Schematic representation of a Partitioned QCA.

operator from eq B.18 is applied generating a new set of such composite states. Equation B.20 shows the procedure of decoding three neighbor cells into an intermediary state and calculating a new composite cell state.

$$\begin{aligned}
 &|101\rangle_h + |011\rangle_i + |100\rangle_j \rightarrow \\
 &\rightarrow U|\tau_h(1) = 1\tau_i(0) = 1\tau_j(-1) = 1\rangle \\
 &\rightarrow |010\rangle \\
 &\rightarrow |\tau_g(1) \tau_h(0) = 1 \tau_i(-1) = 0\rangle_h \\
 &\quad + |\tau_h(1) = 1 \tau_i(0) = 1 \tau_j(-1) = 0\rangle_i \\
 &\quad + |\tau_i(1) = 0 \tau_j(0) = 1 \tau_k(-1)\rangle_j
 \end{aligned} \tag{B.20}$$

with $\tau_h(1)$ being a function allowing to select a value of a qubit from a single composed cell state given the index such as $l \in \{-1, 0, 1\}$. For instance $\tau_h(1)$ returns the rightmost value of the qubit from the composite state of the cell at index h.

□

B.0.3 Quantum Finite Automata and Quantum State Machines

The classical finite automaton(FA) or finite state machine (FSM), is the feasible counterpart of the Turing Machine (TM). While the TM requires an

infinite-size memory, the FSM has a finite memory.

Definition B.0.13 Deterministic 1FSM

A Deterministic FSM is defined by $M = (Q, \Lambda, q_0, Q_a, \delta)$ with Q being the finite set of states of the machine, Λ is the input alphabet, q_0 the initial state, Q_a the set of accepting states and δ the state transition function. The transition function performs a mapping $\delta : Q \times \Lambda \rightarrow Q$ such that for each state q and input symbol λ the next state is given by $\delta(q, \lambda)$. $\square$

Example B.0.3.1 1FSM

For example, let machine M be defined by $Q = \{q_0, q_1, q_2\}$, $q_0 = q_0$, $Q_a = \{q_3\}$, $\Lambda = \{0, 1\}$ and the transition function defined by the Moore state diagram in Figure B.5.

Figure B.5: State diagram of a 1FSM machine. Observe the implicit state transition from q_2 to q_0 for all possible inputs. The state q_3 represents a trap state (rejecting state).

The Moore Machine M , starts in the state q_0 and remains there until either the 1 is read or the end tape symbol $\$$ is read (if the terminal symbol $\$$ is read the machine goes to the accepting state q_2 (generates output 1) and recognizes sequences from the set $\{(10)^*\}$. The machine remains in the state q_1 until either a symbol 0 is read (in which case it returns to the initial state q_0), or the input 1 is read in which case it goes to the accepting state q_2 , generates 1 on output and recognizes the sequence $\{1(01)^*1\}$.

In Logic Design and Synthesis the FSM models are categorized to two main known models: Mealy machine and Moore Machine. The main difference between these models is in the way how the output is generated. In Moore

FSM the output depends only on the current state, while in the Mealy machine alternative the output depends directly on the input and on the current state.

Definition B.0.14 Mealy Output

Let $A = \{Q, \Lambda, Q_0, \delta, g\}$ be an Mealy FSM with Q being the set of states, Λ being the input and output alphabet, Q_0 is the set of initial states, $\delta : Q \times \Lambda \rightarrow Q$ the state transition function and $g : Q \times \Lambda \rightarrow \Lambda$ being a Mealy output function. $\square$

Example B.0.3.2

Let Machine A be defined according to Figure B.6. A Moore functional equiv-

Figure B.6: State diagram of a Mealy FSM machine. Observe notation: 'Input symbol/ Output value' on each transition.

alent can be constructed easily because each Mealy machine has a functional Moore equivalent such that $Q_o = Q_m \times \Lambda_m$. In this case a minimized Moore machine is shown in Figure B.7. In other words, each Mealy machine can be built as a Moore machine provided a state is created for each unique state/input/transition combination in the Mealy machine. $\square$

So far we introduced the QFSM and in particular the 1QFSM that represents a machine reading the input tape in a left-right direction. A natural generalization is the 2FSM that is allowed to move on the input tape in a manner similar to the Turing machine. A Schematic representation is shown in Figure B.8.

Definition B.0.15 Deterministic 2FSM/FA

A Deterministic two-way Finite State Machine is defined by $M = (Q, \Lambda, q_0, Q_a, \delta)$ with Q being the finite set of states of the machine, Λ is the input alphabet, q_0

Figure B.7: State diagram of a Moore FSM machine equivalent to the Mealy machine from Figure B.6. Observe notation: 'Input symbol' on each transition and output specification in the state if different than 0.

Figure B.8: Schematic representation of the 1-way FSM (Top) and a 2-way FSM (Bottom). On the leftmost extremity of the tape is the start marker # and the end marker \$ indicates the end of the input string.

the initial state, Q_a the set of accepting states and δ the state transition function. The transition function performs a mapping $\delta : Q \times \Lambda \rightarrow Q \times \{\leftarrow, \downarrow, \rightarrow\}$ such that for each state q and input symbol λ the next state is given by $(q, \lambda) \times D$. The function D represents the movement generated by the head as a function of the input symbol and state. $\square$

Example B.0.3.3 2FSM

Consider the machine from Figure B.5, and assume that in this case it is a 2FSM machine. The machine specification will now change only with respect

Figure B.9: State diagram of a NDFSMT Moore machine.

to the head movement, and thus we can define:

$$\begin{aligned}
 D(q_0) &= \rightarrow \\
 D(q_1) &= \rightarrow \\
 D(q_2) &= \leftarrow
 \end{aligned}
 \tag{B.21}$$

□

Beside the deterministic machines, there also exists the probabilistic FSM (PFSM) and non-deterministic FSM (NDFSMT). The main differences between these machines and the deterministic models is the transition function.

Definition B.0.16 Non-deterministic FSM

A NDFSMT is a finite state machine specified as in Definition B.0.13 with the difference that the state transition function now takes the form of:

$$\delta : Q \times \Lambda \rightarrow \{0, 1\}^{|Q|} \tag{B.22}$$

□

Example B.0.3.4 NDFSMT

Let Figure B.9 be the state diagram of a NDFSMT. By inspection it can be seen that in states q_0 and q_1 the non-deterministic paths are for input 1 and input 0 respectively. In details one can see that for state q_0 and for input 1 two possible next state transitions are available; q_1 and q_2 . Similarly for the state q_1 and for input 0, the possible next states are q_1 and q_0 . □

Figure B.10: Two Examples of PFSM

Definition B.0.17 Probabilistic FSM

A PFSM is a finite state machine specified as in Definition B.0.13 with the difference that the state transition function now takes the form of:

$$\delta : Q \times \Lambda \rightarrow \{0, 1\}^{|Q|} \quad (\text{B.23})$$

Moreover, the local probability condition must be satisfied such that for each state $q \in Q$ and $\lambda \in \Lambda$

$$\sum_{q' \in Q} \delta(q, \lambda, q') = 1 \quad (\text{B.24})$$

is valid. □

Example B.0.3.5 PFSM

To obtain an autonomous PFSM it is enough to modify the transition function δ of the previously introduced FSM's. Figure B.10a is the state diagram of an autonomous PFSM (its next state depends solely on its current state) and Figure B.10b shows a diagram of a standard PFSM. Unlike in the NDFSM case, the PFSM state transitions and output generation are weighted by a probability. For every input symbol in every state the sum of probabilities of outgoing arrows must be 1 (eq. B.24). □

In this case it is also possible to simply describe the behavior of the autonomous PFSM by a state transition matrix.

Figure B.11: Diagrams representing the PFSMs from Figure B.10

$$V = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & 0 \end{pmatrix} \quad (\text{B.25})$$

Equation B.25 shows an example of such a matrix with the output being read from the left side. The states are represented as orthonormal vectors

$$q_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, q_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, q_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad (\text{B.26})$$

and the computation is done by applying the state transform to the current state. Thus starting in state

$$\delta(q_0) = \frac{1}{2}q_0 + \frac{1}{4}q_1 + \frac{1}{4}q_2 \quad (\text{B.27})$$

one can end up in the same state with probability $\frac{1}{2}$ and with probability $\frac{1}{4}$ in either the state q_1 or state q_2 .

Quantum Finite State Machine (QFSM) is also a natural extension of classical (probabilistic) FSM's. Two main types of QFSM are well described: One-way QFSM (1QFSM) and two-way QFSM (2QFSM). As will be illustrated and explained the 1QFSM accepts classical input and measures its quantum memory after each operation (Figure B.12).

In contrast to that, the 2QFSM is designed to operate on quantum input data

Figure B.12: Schematic representation of a 1QFSM; The machine reads input, performs Unitary transform on the quantum state, measures the quantum state and the output state.

(allowing to put the reading head in superposition with the input tape) and the measurement is done only at the end of a whole process.

Definition B.0.18

Quantum State Machine - a QFSM is a tuple $\Gamma = \{Q, \Lambda, q_0, Q_{ac}, Q_{rj}, \delta\}$, where Q is a finite set of states, σ is the input alphabet, δ is the transition function. The states $q_0 \in Q'$, $Q_{ac} \subset Q$ and $Q_{rj} \subset Q$ are the initial states, the set of accepting states and the set of rejected states, respectively. $\square$

In the case of the 1QFSM, the state transition function δ maps:

$$Q \times \Theta \times Q \rightarrow C \quad (\text{B.28})$$

with $\Theta = \Lambda \cup \{\#, \$\}$ ³ and with $\{\#, \$\} \notin \Lambda$. In other words the machine action maps the set of machine states and the set of input symbols into the set complex of machine next states. The computation of such machine is required to be done using unitary operators and is performed on the basis set B^q using unitary operators U_θ , $\theta \in \Theta$

$$U_\theta = \sum_{q' \in Q} \delta(q, \theta, q') |q'\rangle \quad (\text{B.29})$$

The probability of measuring the machine state $|q'\rangle$ is then given by: $p(|q'\rangle) = \delta(q, \theta, q')^* \delta(q, \theta, q')$. The 1QFSM operates on real data (represented as values of basis states of the reading system) and after each step of δ the machine's

³ # represents the start of a input string and \$ represent the symbol indicating that the data word is terminated.

quantum state is measured [KW97, Wat95a] and the current state is evaluated within the orthonormal decomposition of the observable space:

$$G = G^a \otimes G^r \otimes G^n \quad (\text{B.30})$$

where $|q'\rangle\langle q'|$ is accepted, rejected or computed (eq. B.29) when measured in G^a, G^r, G^n subspaces, respectively. Thus for a given string to be processed and prior to the whole process termination (string either accepted or rejected), the overall processing can be represented as:

$$MU_{\theta_n} MU_{\theta_{n-1}} MU_{\theta_{n-2}} \dots MU_{\theta_3} MU_{\theta_2} MU_{\theta_1} |q_0\rangle \quad (\text{B.31})$$

with MU_{θ_n} being the application of the U_{θ_n} operator to the current state and creating the configuration $U_{\theta_n}|q\rangle$ followed by the measurement of the current state M (projecting the state into G). The 1QFSM was proven to be less powerful or equally powerful to its classical counterpart 1FSM [Gru99, KW97]

The above described 1QFSM is also called the measure-many quantum finite automaton [KW97]. A model called measure-once quantum finite automata was also introduced and studied by Moore [MC00]. The measure-many 1QFSM is similar to the concepts of the 2QFSM presented below.

Example B.0.3.6 1QFSM

Let $Q = \{|q_0\rangle, |q_1\rangle\}$ be two possible states (omitting the accepting and rejecting states) of a single-qubit machine M and with transition functions specified by the transitions defined in eq. B.32 corresponding to the state diagram in Figure B.13a.

$$\begin{aligned} V_{\#}|q_i\rangle &= |q_0\rangle & V_0|q_0\rangle &= \frac{1}{\sqrt{2}}|q_0\rangle + \frac{1}{\sqrt{2}}|q_1\rangle \\ V_{\$}|q_0\rangle &= |q_{ac}\rangle & V_0|q_1\rangle &= \frac{1}{\sqrt{2}}|q_0\rangle - \frac{1}{\sqrt{2}}|q_1\rangle \\ V_{\$}|q_1\rangle &= |q_{rj}\rangle & V_1|q_0\rangle &= |q_0\rangle \\ & & V_1|q_1\rangle &= -|q_1\rangle \end{aligned} \quad (\text{B.32})$$

The machine M, specified in eq. B.32 represents a state machine that uses the H gate when the input is 0 ($V_0 = H$) and the Pauli-Z rotation gate when the input is 1 ($V_1 = Z$). Observe that machine M would have different behavior for

Figure B.13: (a) State transition diagram for the 1QFSM defined by the transition function B.32, (b) the representation of the QFSM using quantum multiplexers. Observe two control outputs $|q\rangle$ specifying the machine action/states and the input symbols selecting the appropriate unitary transform V_i for $i \in \{\#, \$, 0, 1\}$.

measure-once and measure-many implementation. In the measure-many case, the machine generates a quantum coin-flip while receiving input 0 and while receiving input 1 the Pauli-Z rotation is applied. Observe in the measure-once case, that for example for the string input $\theta = "010"$ the many-measure machine will implement a NOT using $[H][Z][H]$. $\square$

Note that in this approach to QFSM each input is represented by a unitary transform that can be seen as shown in Figure B.13. The 2QFSM operates on a similar principle as the 1QFSM model but with the main difference being the application of the measurement. This is explained below with the 2QFSM definition, as illustrated in Figure B.14.

Definition B.0.19 2QFSM

A 2QFSM is defined as $M = (Q, \Lambda, q_0, Q_a, Q_r, \delta)$ with Q_a and Q_r being the set of accepting and rejecting states respectively. The state transition function δ

Figure B.14: Schematics representing the difference between the 1QFSM and 2QFSM. On the top, the 1QFSM - for each input character read from left to right from the tape, a unitary transform U is applied on the state and the state is measured. On the bottom, the 2QFSM moves on the input tape left and right, the unitary transform U is applied on the state and only once the computation is terminated the final state is observed/measured.

mapping the from the set of states and inputs to the set of machine states as well as to the head movement:

$$Q \times \Lambda \times Q \times \{\leftarrow, \downarrow, \rightarrow\} \rightarrow \mathbb{C}_{[0,1]} \quad (\text{B.33})$$

□

Using this definition and eq. B.33 allows to put the machine M in a more complex quantum state (the head can be in superposition with the tape - exploring multiple paths at once). Therefore the two-way QFSM was shown to be more powerful than classical the 2FSM [Wat97] by allowing to recognize some non-regular languages as for instance the language $L = \{a^n b^n | n > 0\}$.

The most standard model of 2QFSM is called the *simple 2QFSM* and is defined by a unitary state transition function δ defined in eq. B.34.

$$\delta(q, \lambda, q', d) = \begin{cases} \langle q | U_\lambda | q' \rangle, & \text{if } D(q) = d \\ 0, & \text{otherwise} \end{cases} \quad (\text{B.34})$$

Observe that this definition entails the fact that each state is associated with a single well-defined head movement such that $D(q) = d$. This is also the reason that this 2QFSM is also called a *unidirectional 2QFSM*.

The transition function δ is ensured to be unitary if it satisfies the following conditions:

- the sum

$$\sum_{q_0, d} \delta^*(q_1, \lambda_1, q_0, d) \cdot \delta(q_2, \lambda_2, q_0, d) = \sum_{q_0, d} \langle q | U_\lambda^\dagger | q' \rangle \langle q' | U_\lambda | q \rangle = 1 \quad (\text{B.35})$$

if $q_1 = q_2$ and 0 otherwise, implying that every two distinct states of the 2QFSM must be orthonormal

- and the three separability conditions;

$$- \sum_{q_0} \delta^*(q_1, \lambda_1, q_0, \rightarrow) \cdot \delta(q_2, \lambda_2, q_0, \downarrow) = 0$$

$$- \sum_{q_0} \delta^*(q_1, \lambda_1, q_0, \downarrow) \cdot \delta(q_2, \lambda_2, q_0, \leftarrow) = 0$$

$$- \sum_{q_0} \delta^*(q_1, \lambda_1, q_0, \rightarrow) \cdot \delta(q_2, \lambda_2, q_0, \leftarrow) = 0$$

- these three conditions imply that each state of the 2QFSM must be accessible in a unique way; only under such conditions the overall process of computation using a 2QFSM can be written as a unitary matrix of a well-formed quantum system.

To describe the evolution of the 2QFSM the computation can be expressed as a function of configuration. Thus let:

Definition B.0.20 2QFSM configuration

A configuration of a 2QFSM is a tuple $\langle q, l \rangle$ with q being the state of the configuration and l being the position of the head. $\square$

The set C_m (of size $(m + 2)^{\otimes 2^n}$) represents all possible configurations of M (with $|Q| = 2^n$ for an input of size m). The state transition function can be seen as an unitary transformation applied with respect to the configuration of the machine:

$$U_l^\delta(q, l) = \sum_{q', d} \delta|q, w_x(l), q', d|q', (k + \mu(d)) \bmod (n + 2) \rangle \quad (\text{B.36})$$

where $w_x(l)$ is the symbol at the l^{th} position in the $w_x = \#x\$$ (Tape) and d defined as $D(q') = d$ with $D : Q \rightarrow \{\leftarrow, \downarrow, \rightarrow\}$ (being the result of moving the head on the tape)[Gru99].

Example B.0.3.7 2QFSM for $\{0^n 1^n | n \geq 1\}$

Let's look at the machine introduced by Watrous [Wat97], that recognizes the language $L = 0^n 1^n | n \geq 1$. This machine functionality can be presented as follows:

- - first move from the starting position at the left end of the tape, to the right side until reaching the terminal symbol. During this phase the machine checks whether the word is of the form $\{0^n 1^m | n, m \geq 0\}$ (which is a regular language and can be recognized by classical FSM). Such a machine is described by $M = \{Q, \Lambda, q_0, q_{ac}, q_{rj}, \delta\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $q_{ac} = \{q_3\}$, $q_{rj} = \{q_4\}$ and the transition function δ is specified by the transitions below in equation B.37. A different notation is used in order to allow easy understanding of the unitary operators used. This means that as to each symbol corresponds a particular notation instead of $\delta(q, \lambda) \rightarrow q'$ a single state transition is expressed in the form of equation such as $V_\#|q_0\rangle = |q_0\rangle$. Again this QFSM can be represented as shown in Figure B.13b.

$$\begin{array}{llll} V_\#|q_0\rangle = |q_0\rangle & V_0|q_0\rangle = |q_0\rangle & V_1|q_0\rangle = |q_1\rangle & V_\$|q_0\rangle = |q_1\rangle \\ V_\#|q_1\rangle = |q_2\rangle & V_0|q_1\rangle = |q_2\rangle & V_1|q_1\rangle = |q_0\rangle & V_\$|q_1\rangle = |q_0\rangle \\ V_\#|q_2\rangle = |q_4\rangle & V_0|q_2\rangle = |q_4\rangle & V_1|q_2\rangle = |q_2\rangle & V_\$|q_2\rangle = |q_3\rangle \\ V_\#|q_3\rangle = |q_3\rangle & V_0|q_3\rangle = |q_3\rangle & V_1|q_3\rangle = |q_3\rangle & V_\$|q_3\rangle = |q_2\rangle \\ V_\#|q_4\rangle = |q_1\rangle & V_0|q_4\rangle = |q_1\rangle & V_1|q_4\rangle = |q_4\rangle & V_\$|q_4\rangle = |q_4\rangle \end{array} \quad (\text{B.37})$$

The head movement function is specified as follows:

$$D|q_0\rangle = \rightarrow \quad D|q_1\rangle = \leftarrow \quad D|q_2\rangle = \rightarrow \quad D|q_3\rangle = \downarrow \quad D|q_4\rangle = \downarrow \quad (\text{B.38})$$

This machine, as it can be verified is Unitary (each operator V is unitary and the machine satisfies the separability conditions as defined above) and will accept any string $s = \{0^n 1^m | n, m \geq 0\}$ with probability 1.

- - on the rightmost marker the machine generates a superposition of states representing different possible paths for the head on the tape. Assume that at the end marker (the rightmost end of the tape) the machine is in state q_3 , then

$$V_{\$}|q_3\rangle = \frac{1}{\sqrt{n}} \sum_i^n |l_{j,0}\rangle \quad (\text{B.39})$$

- - the machine sends all (superposed) heads in different states $|l_{j,0}\rangle$ in parallel back in such a way that if the length of their respective paths that the heads will travel is different, the heads in various superposed states will arrive at different times to the destination. This can be described by the following state transitions:

$$\begin{aligned} V_0|l_{j,0}\rangle &= |l_{j,j}\rangle, 1 \leq j \leq n \\ V_0|l_{j,k}\rangle &= |l_{j,k-1}\rangle, 1 \leq j \leq n, 1 \leq k \leq j \end{aligned} \quad (\text{B.40})$$

and

$$\begin{aligned} V_1|l_{j,0}\rangle &= |l_{j,n-j+1}\rangle, 1 \leq j \leq n \\ V_1|l_{j,k}\rangle &= |l_{j,k-1}\rangle, 1 \leq k \leq j \leq n \end{aligned} \quad (\text{B.41})$$

- - perform Quantum Fourier Transform, that allows to extract the result. All heads arrived to the leftmost starting marker are in state $|l_{j,0}\rangle$, thus

$$V_{\#}|l_{j,0}\rangle = \frac{1}{\sqrt{n}} \sum_{o=1}^n e^{\frac{2\pi i j o}{n}} |t_o\rangle, 1 \leq j \leq n \quad (\text{B.42})$$

performs a QFT on each head in parallel.

- - measure the state of the 2QFSM. If all the QFTs occur simultaneously

, the general form of the state of M is $\frac{1}{n} \sum_{j=1}^n \sum_{o=1}^n e^{\frac{2\pi i j o}{n}} |t_j\rangle$, $1 \leq j \leq n$, and thus sums to $|s_n\rangle$ and the result is obtained with probability 1. In the case when the splitting of the final state does not occur simultaneously (none of two paths are analyzed in the same time) when M is measured, then the probability of rejecting the result is thus $1 - \frac{1}{n}$.

□

The 2QFSM was extended by Ambainis and Watrous [AW02] to an even more powerful concept called the 2QCFA (2-way Finite automata with quantum and classical states) allowing to recognize some non regular languages (such as $L_{nr} = \{a^n b^n | n \in N\}$) as well as palindromes ($L_{pl} = \{x \in a, b^* | x = x^R\}$). This machine has a mixed state with two state variables. The first state variable being classical (interacting with the tape and the reading head - thus accepting real/boolean information). The second state variable is used as the computation memory and allows to obtain quantum effects improving its performance. In particular, in the original model the quantum bit was used as a quantum counter.

Definition B.0.21 2QCFSM

The 2QCFSM is a tuple $M = (Q, S, \Upsilon, q_0, s_0, S_{acc}, S_{rej}, \tau, \delta)$ with Q and S being respectively the set of quantum and classical states, Υ being a finite alphabet, $q_0 \in Q$ and $s_0 \in S$ are the initial quantum and classical state, S_{acc} and S_{rej} are the set of accepting and rejecting classical states.

The function $\tau(s, v)$ (with $(s, v) \in S \setminus (S_{acc} \cup S_{rej}) \times \Upsilon \cup \{\$, \#\}$ being a state and a character from the finite alphabet $\Upsilon \cup \{\$, \#\}$) describes the evolution of the quantum portion of the state register by allowing two quantum operations: the unitary matrix operator and the measurement operator. The function δ controls the classical part of the machine such that

- if $\tau(s, v)$ is a unitary transform then $\delta(s, v) \in S \times \{-1, 0, 1\}$ specifying the new classical state and the head movement
- if $\tau(s, v)$ is a measurement then $\delta(s, v)$ is just a transition function mapping the measured quantum state to $S \times \{-1, 0, 1\}$.

The functional diagram of an example QCFA is shown in Figure B.15. □

Figure B.15: The schematic of the 2QCFSM; QL and CL represents quantum logic (function τ) and classical logic (function δ), respectively. Q represents the quantum state register and C represents the register with classical state. Remark that in this case the measurement is represented in the QL block. Thus the only time when the quantum state is used is when it is measured.

B.0.4 Quantum Turing machine

Quantum Turing Machine (QTM) - is a natural extension of the classical Turing Machine similar to the probabilistic Turing machine, and with the main difference that in the QTM the transition probabilities are complex numbers.

Definition B.0.22 QTM

A QTM is defined by a five-tuple $M = \{\Sigma, Q, q_o, q_f, \delta\}$, where Σ is an infinite tape, Q is the set of states, q_o and q_f are the initial state and the final state respectively, symbol δ is the state transition function defined by:

$$\delta : (Q \times \Sigma \times \Sigma \times Q \times (\leftarrow, \downarrow, \rightarrow)) \rightarrow C_{[0,1]} \quad (\text{B.43})$$

The transition function maps the machine into a new complex state as well as to an action to be done on the tape. $\square$

Similarly to previous devices, a configuration of a QTM is a triplet $\{\sigma, i, q\}$, where σ is the symbol (content) on the tape. It specifies the position of the head (the symbol read/written) and $q \in Q$ the current state of machine. The transition function δ is a mapping $m : C \times C \rightarrow C$ such that for $c_1, c_2 \in C$, $m(c_1, c_2)$ is the amplitude of the transition from the basis state c_1 to c_2 . The time evolution behavior of the QTM is then for arbitrary state a mapping $V : H \rightarrow H$ such that $V_M|c\rangle = \sum_{c \in C_M} \alpha_c U_M|c\rangle$.

A QTM is unitary if the evolution (function δ) is unitary and for it must be true that:

•

$$\sum_{\{q_0, \sigma_0, d\} \in Q \times \Sigma \times D} |\delta(q_0, \sigma_0, q_1, \sigma_1, d)|^2 = 1 \quad (\text{B.44})$$

(with $q_1 \in Q$ and $\sigma_1 \in \Sigma$), specifying the local probability condition (each state must be accessible),

•

$$\sum_{\{q_0, \sigma_0, d\} \in Q \times \Sigma \times D} \delta^*(q_0, \sigma_0, q_2, \sigma_2, d) \delta(q_0, \sigma_0, q_1, \sigma_1, d) = 0 \quad (\text{B.45})$$

for $(q_1, \sigma_1) \neq (q_2, \sigma_2)$ specifying the orthonormality condition assuming same head movement (in a given state q , with input symbol σ and a particular head movement only one state must be reachable).

•

$$\sum_{\{q_0, \sigma_0\} \in Q \times \Sigma} \delta^*(q_0, \sigma_0, q_2, \sigma_2, d_2) \delta(q_0, \sigma_0, q_1, \sigma_1, d_1) = 0 \quad (\text{B.46})$$

specifies that there are no two different states of M $(q_1, \sigma_1, d_1) \neq (q_2, \sigma_2, d_2)$ that would lead to the same configuration.

- Finally the same state $q \in Q$ of M , cannot be generated from two different configurations and being part of two different configuration of the machine M :

$$\sum_{\{q_0, \sigma_0\} \in Q \times \Sigma} \delta^*(q_0, \sigma_0, q_2, \sigma_2, d_2) \delta(q_0, \sigma_0, q_1, \sigma_1, d_1) = 0 \quad (\text{B.47})$$

B.0.5 Quantum Robots

It was Benioff [Ben98a, Ben98b] who proposed the quantum robot as a device to search quantum spaces [Ben02]. In his approach the robot was defined as a QTM and extended to QTM with multiple internal tapes.

Definition B.0.23 Quantum Robot

Quantum robot is a 3-tuple $R = \{M, O, c\}$ with $M = \{\Sigma, Q, q_o, q_f, \delta\}$ a QTM, O being a finite state output system and c is a control qubit. $\square$

The Quantum robot is equipped with an on board Quantum Computer (QC) and all required auxiliary devices such as measurement aperture, the output

system O and the control bit c . The robot environment is quantum and is defined in terms of sub-spatial regions such as $E = \sum_{x,y,z} E_x |b\rangle$ spanned by a set of orthonormal bases defined by $\sum_x |x\rangle$.

The robot executes discrete actions in a discrete quantum environment/space in such a way that if the robot position state is given by $|x_R\rangle$ then moving by three steps to the right will result in physical movement with respect to x such that $|x_R\rangle = |x_R + 3\rangle$ represents the translation in space. As a consequence of the robot being defined as a QTM, the robot can operate on a superposition of environments such as: $|\mathbf{x}\rangle = \otimes_{j=1}^n |x_j\rangle$ and thus a movement in such a superposed quantum environments $|\rho\rangle = \sum_x c_x |x\rangle$ will be performed in all dimensions such that: $\sum_x c_x |x + 3\rangle$. Thus a robot being in a superposition with its environment can, similarly to parallel computation, explore many different paths simultaneously.

The Quantum robot executes reversible tasks (physical reversibility) such as movement in space, however it must also be able to perform non reversible tasks such as for example cleaning of a spatial region or measurement of a qubit [Ben98a]. For example, the 'clean up region' is described by $\sum_x c_x |x\rangle |0\rangle_{buf} \rightarrow |y\rangle \sum_x c_x |x\rangle_{buf}$ with $|x\rangle$ being the space region to clean and conservation of matter dictates that the removed 'stuff' from region $|x\rangle$ is to be stored/kept in the buffer state $|0\rangle_{buf}$. This action requires a copy operation such as $\sum_x c_x |x\rangle |0\rangle_{buf} \rightarrow \sum_x c_x |x\rangle |x\rangle_{buf}$, with $|x\rangle_{buf}$ is the buffer qubit holding initially being set to $|0\rangle$ and $|x\rangle$ is the state of the region R . The spatial regions R can be defined with respect to a measured particle p such as $|E_x\rangle = |x\rangle_p |E\rangle_{\neq p}$ with $|E_x\rangle$ being the state of the environment with the particle at position x . A schematic example of such machine can be seen in Figure B.16

The robot operation is described in two separate phases: the computation phase P_c and the action phase P_a . The purpose of this separation is to allow the robot to evolve separately of the environment as well as to allow the robot to act on the environment while remaining in the same state. Let $|E\rangle = \otimes_{j=1}^n |x_j f_j\rangle$ be an environment with n particles (for the purpose of explanation it is possible to think of the environment as a quantum buffer and the values indicate the presence or absence of an obstacle (a particle) on coordinates x), where $|x_j\rangle$ is the position index of the particle and $|f_j\rangle$ is its value. With respect to previous considerations, let $N(\mathbf{x})$ be the environmental neighborhood of the robot and the environment can be rewritten as $|E\rangle = |E\rangle_{N(\mathbf{x})} |E\rangle_{\neq N(\mathbf{x})}$.

The requirements of the two phases P_c and P_a can be expressed with respect to

Figure B.16: A schematic representation of a quantum robot by Benioff. L_1 is a one dimensional space (with head h_1), L_2 is a finite tape of the on-board QTM (with head h_2), o is the output system and c is the control qubit. On the left the robot is shown using the QTM representation, while on the right he robot is represented as a quantum circuit, with the logic representing the robot control (shown as R).

E such as $\langle \mathbf{x}, E | P_c | \mathbf{x}, E \rangle = \langle E_{N(\mathbf{x})} | P_c | E_{N(\mathbf{x})} \rangle$, with $\langle E_{\neq N(\mathbf{x})} | E_{\neq N(\mathbf{x})} \rangle = 1$ for the computation phase. Similarly, for the action phase P_a must be independent (not changing) the state of the quantum computer on-board as well as the state of the environment. This means that while the quantum robot does quantum computation, its quantum environment must remain unchanged by its actions. It can be expressed as $\langle \mathbf{x}' E' | P_a | \mathbf{x}' E' \rangle = \langle E'_{\neq N(\mathbf{x}', \mathbf{x})} | E'_{\neq N(\mathbf{x}', \mathbf{x})} \rangle \times \langle \mathbf{x}' E'_{N(\mathbf{x}', \mathbf{x})} | P_a | \mathbf{x} E_{N(\mathbf{x}', \mathbf{x})} \rangle$ where $E_{N(\mathbf{x}', \mathbf{x})}$ and $E_{\neq N(\mathbf{x}', \mathbf{x})}$ are the spatial regions inside and outside of the robot neighborhood.

Another way of describing the quantum robots is using physical approach based on Feynman path integrals [FH65]. Assume $\Phi(0)$ and $\Phi(n)$ are the initial state and the state after n tasks steps of a robot respectively. For a quantum robotic system specified by the state $|\rho, o, c, x, E\rangle$ after n phases, with q_{comp} represents the quantum computer state qubit, the expression of the

final state $\Phi(n)$ becomes:

$$\langle \Phi(n) \rangle = \sum_{\rho_1, o_1, c_1, \mathbf{x}_1, E_1} \langle \rho, o, c, \mathbf{x}, E | P^n | \rho_1, o_1, c_1, \mathbf{x}_1, E_1 \rangle \times \langle \rho_1, o_1, c_1, \mathbf{x}_1, E_1 | \Phi(0) \rangle \quad (\text{B.48})$$

The matrix element $\langle \rho, o, c, \mathbf{x}, E | P^n | \rho_1, o_1, c_1, \mathbf{x}_1, E_1 \rangle$ can be re-expressed as a sum over all possible paths from the initial state $\Phi(0)$ to the final state $\Phi(n)$. For this let $|\rho_1, o_1, c_1, \mathbf{x}_1, E_1\rangle$ be expressed as $|w, c\rangle$ and

$$\begin{aligned} \langle w, c | P^n | w_1, c_1 \rangle &= \sum_{w_2, i_1, \dots, w_n, i_n} \langle w, c | T | w_n, c_n \rangle \langle w_n, c_n | T | w_{n-1}, c_{n-1} \rangle \\ &\dots \langle w_2, c_2 | T | w_1, c_1 \rangle \quad (\text{B.49}) \end{aligned}$$

be the expansion in a complete set of states between each P. Following [FH65] the equation B.49 can be rewritten as the sum over paths of the states $\{|w, c\rangle\}$:

$$\begin{aligned} \langle w, c | P^n | w_1, c_1 \rangle &= \sum_{r \in R^{n+1}} \langle r_{n+1} | P | r_n \rangle \langle r_n | T | r_{n-1} \rangle \\ &\dots \langle r_2 | T | r_1 \rangle \langle r_{n+1} | T | w, c \rangle \langle r_1 | T | w_1, c_1 \rangle \quad (\text{B.50}) \end{aligned}$$

The robot behavior is functionally controlled by the qubit c that separates the two robot phases: action and computation. Each phase is given a projection operator P_a and P_c respectively. A robotic behavior, now can be seen as a sequence of alterations at discrete time steps t and on a limited space of size n . The total number of phases that the robot goes through is also $n \sum_{i=1}^n j_i = n$. This alteration between the two actions is shown in eq. B.51, with O_0 being the projector for the control state c for the value of 0 and inversely for O_1 being the projector for the control state c for the value of 1.

$$P^n = (O_0 + O_1)T(O_0 + O_1) \dots (O_0 + O_1)T(O_0 + O_1) \quad (\text{B.51})$$

C CYNTHIA, AN INTERACTIVE ROBOTIC SYSTEM

C.1 Definitions

In the functional description of the robot we use multiple references to terms related to emotion. This section starts by giving some definitions in order to completely define the terminology used in the robot.

- *Dynamic Mood Mapping* is the state to emotion mapping based on the current state, the environment state and the robot complexity. Define $(E, S, \Psi, E_0, \Lambda)$ as the
- *Robotic emotion* behavior or a state of a Humanoid robotic agent that can be identified and distinguished by either evaluation criteria or by human evaluator.
- *Emotional State*. There are two levels of the emotional state in the robot: the state of the software module (local) and the state of the whole robot (global). The local emotional state is the representation of the software module or device itself. It represents the device ability to function and react properly to the environment. The global emotional state is based on the interaction of the local emotional modules. These various modules are the dynamical expression of the emotions in the robot. Here, the emotional state is denoted by Ψ for the state of the whole robot. The emotional state of various robot's parts is denoted by Ψ_n , where n is the index of the robot part. Individual variations to this notation will be explained along with examples.
- *Hedonic Function* is the emotional function based on principles of well-being, equilibrium and reward maximization. The core of the emotional algorithm is based on two functions that are altering both the energy and the energy function parameters. We refer to these functions as the energy function E and the strategy function S . For formal description these are combined into the emotional function Ω such as $\Omega(\Psi) = S(E(\Psi))$.
- *Robotic Mood* is the state of the robot that can be explicitly distinguished and described as a emotional behavior in time. In fact, the robotic

mood is a part of the evaluation of the robot; it represents a successful identification of a robotic behavior by a human operator in a way similar to that of the operator would recognize moods in humans.

- *Robotic Behavior* is the dynamical expression of the robot. The links between mood and behavior are however not a one-to-one mapping. For example, in the case of more elaborate strategy such as lying or pretending.

In this chapter the physical implementation of the Cynthea robot is presented. The robot is described by groups of functional units. The framework of experimentation is explained as well as the conceptual view of integrating the robot with the previously proposed logic synthesis methods. We introduce the CRL language that is used as the framework to link the behavior of the robotic elements together. The CRL language allows behavioral control to be constructed from bottom-up (functional units emerge together as a single robot) also at the high-level robotic behaviors level.

The overall idea of the Cynthea framework is to test how well small quantum circuits/ robotic behaviors can help to improve the standard approaches to behavioral robotics. The robot is separated to two layers: Cognitive (control) and Emotional (individual). The cognitive level is represented by the commands and the actions generated in CRL and given as user scripts for execution. The emotional layer, represents a common layer to all functional units that is controlled by the automated learning/synthesis method.

C.2 Cynthea - the social interactive robotic framework

The Cynthea (Cybernetic Networked Humanoid Emotional Agent) robot is designed to be used as a tool for studying human emotions and to explore robotic emotional behaviors based on human and animal psychology. Similar to a natural world, this robot was built as a metaphorical model of a human that has a body and a brain. Or, in other words, the robot has one cognitive part (command processing and symbolic knowledge) and one emotional part (behavior, body state, body feedback). The cognitive or symbolic part takes care of all command processing as well as of all cause-to-effect phenomena. The emotional part does not process data in the sense of robot Input/Output (I/O) mapping but rather only alters robot's parameters, settings and states based solely on the robot state itself.

This way, the robot allows to express a variety of behaviors not only directly based on inputs and outputs but also on some internal hidden state, that can cause the robot to behave in a novel and unexpected manner. As already introduced, one of the goals of the robot is to provide a platform to generate a set of examples for our machine learning approach.

C.3 Learning and Adaptation

The learning process is implemented in the robot using the GA presented in Chapter 4. The Robot is used as pattern generator for incompletely specified behaviors. It generates sets of input-output symbolic pairs that are called examples of behaviors. The GA extrapolates the specifications into a completely defined quantum functions represented as quantum circuit. As will be described later in this chapter, the robot is built from small networked functional modules that each can be configured to express a particular behavior. This allows to experimentally test the learning approach, because large quantum circuits are not computationally tractable due to the exponential growth in size with additional qubits. However it is possible to synthesize many small-sized quantum circuits that can be then applied to each functional module individually.

Figure C.1: The Evolutionary Quantum-Logic synthesis based Machine Learning Framework.

The overall process of learning can be described by the following steps:

- Initially, the robot behavior is specified by a behavioral script written by a human designer in the CRL language. This script specifies a desired sequence of robot's actions, without considering the inputs. The script specifies only the actions leading towards achieving a desired goal.
- The robot loads the script and executes it. With respect to its current state and its inputs, the robot (software) modifies the behavior into a particular action sequence. The results represent a globally described behavior, with each functional module executing its task given its internal state (hidden state).
- The internal state of each functional module (inputs, states, outputs) is used as an example of robotic behavior. Such behavior is then fed to the GA and re-synthesized in order to observe another behavior with a different observable output (this is shown in Figure C.1)

The CRL language specifies the behavior that does not change without learning. The robot will always respond with some behavior that is a one-to-one mapping to any given CRL command, when no emotions are used. The adaptation of the robot is managed through the dynamic replacement of the internal state assignment functions; the emotions. The learning process in Cynthea robot is in the development stage and has been tested only offline; a set of benchmarks was generated by the robot and then modified by the GA. The new circuit was then inserted into the robot and the behavior executed.

The robot software structure (architecture) can be seen as a two level multi-agent system for robotic control. The emotional plane showed in Figure C.2 represents a separated, parallel to the symbolic level, dimension of the robotic controller, that processes only internal variables and parameters. Unlike in other approaches to emotional robotics [Bre02, OS99], here the emotional level cannot generate any action or activity of the robot by itself. This allows Cynthea to execute and re-execute the same program with its external observable desired behavior, while altering this behavior by its own individual and unique emotional state. This in essence allows to directly observe, compare and gather statistical data about emotional behavior given a precise well formulated action frame. The Cognitive plane processes the actual input to output data and is the main agent generating the robot's actions. The idea behind this concept is that the mapping from inputs to outputs is predetermined by the robot structure, software modules and behavioral modules, and

the emotional level alters the execution patterns and behavioral parameters (such as speed or angle).

Beside the two layer parallelism, the interaction between the two layers is non symmetric. This can be explained as follows. The cognitive layer uses a symbolic language, requiring coding ¹. The language describes input/output actions, similarly to a neural control system. Emotional system is however using a language unknown directly to the cognitive system. This approach intends to simulate the interaction between the chemical system and the electric one in a human. The neural system provides complex encoded signals controlling all activities of our body and of the brain. This also includes the control of organs controlling directly the nervous system itself, such as pineal gland, glial cells, and other hormonal and enzymatic functions.

There is a difference between these two co-systems: the cognitive system controls the robotic system with an explicit language that is directly translated into action and low level commands; the emotional system uses an implicit language, that cannot be directly translated to low level action and cannot be broken into distinct commands.

Thus, the Cognitive Level processes information in the strict sense, while emotion is used to only alter the already existing processes or behavioral and action modules. The parameters and variables that the emotion is build upon are the energy parameters, hedonic functions and self-satisfaction variables (Section C.7); the sole representation of the robot itself. This is explained in more details in Section C.7.

Figure C.2 shows how the two planes, cognitive and emotional, inter-operate. The dashed line represents the emotional pathways and the full line represents the cognitive data path. The input to the emotional level is emotional information (numerical data streams) from the software module itself (described later) or from other emotional elements of the robot. First, the emotional level is processing emotional information (dashed arrows from left to right) as well as it influences the cognitive level (dashed line from top to bottom). It can be noticed that there is no emotional control back from the cognitive plane. The cognitive information is processed through the emotional level to the cognitive level (solid lines) and is out-putted back through the emotional level. This will

¹ a deterministic language allowing to express meaning in given data measured with finite precision, must be finite with finite precision so as it can be understood by another machine with finite deciding possibilities

be explained in more details for each robot function presented in the sequel and also in Section C.7.

The robot function is to execute a command or a set of commands in an individual and unique emotional manner (examples can be seen in Section C.6.1). The idea behind this approach is to search for the answers to questions such as: (1) can we design a well-defined framework allowing to successfully replicate human emotions as expressed in human-robot interaction (verbal and non-verbal) or (2) can we build a robot that will successfully use features of human emotions such as adaptation, innovation and express a unique robotic behavior.

Figure C.2: Two level schematic representation of the Cognitive-Emotional robot structure. The 'emotional plane' represents the non symbolic level of processing, the 'Cognitive plane' represents the symbolic level of processing. The sensory input starts on the 'emotional plane' then is processed by both 'emotional' and 'cognitive' levels in parallel. The output generated by the robot is again a resulting action from both the 'emotional and 'cognitive' planes.

All parts (software components) of the robot are functional modules, implemented in the same way, every module has a functional part and a emotional part. Thus, when the focus is on a sensor, the sensor is able to process input information into signals that the robot understands and can explicitly use. However, the emotional part of every module is not directly controllable by neither the robot nor the user. Because Cynthea system is modular, its emo-

tional state is dependent on its complexity: the more modules are added to the robot and more of them are active, the more the emotional state of the robot can vary and thus become interesting to the observer from the point of view of the behavior and interaction generation.

C.4 Sensors

The robot is equipped with various sensors to sense the external surrounding environment as well as its own body-environment; i.e. extero-sensors, proprio-sensors and intero-sensors. The sensory differentiation in our model (related to the biology of human and animal bodies) is well defined. Despite this fact the robot does not differentiate between external and internal signals by mean of specific sensors. Thus, from the functional point of view all sensors used in the robot are of the same type. The states of all sensors are interpreted by the same mechanisms of reasoning, planning and acting.

The hardware sensors oriented to the exterior (extero-sensors) of the robot are Video Camera, Audio Input, (Incoming)Touch Sensors, Sonar and thermo-sensors. Each sensor is associated with a set of routines allowing to give a modular description of the robot so that every sensor can be enabled, disabled, altered or so.

Figure C.3: The generic sensor module. As all parts of the robot, it is made from two components: emotional and cognitive. The cognitive part is function-specific and the emotional part is similar to all other emotional modules.

The emotional level in the sensor allows to alter the functionality and the

quality of the perceiving process ² and of the perceived data. Thus, given a sensor S, its activity can be in the extreme cases turned off or turned on by the emotion. Also, for example in a vision sensor, between these two extremes, parameters can be altered allowing different levels of granularity for object recognition, various levels of reliability in object tracking, and so on. The type of adjustment that a sensor can be subject to depends on its function and the type of data that it accepts and processes.

Vision sensor, audio sensor and touch sensor are considered as extero-sensors. Thermo-sensor, voltage/power sensor are intero sensors (indicating internal state of the robot). Finally the proprio sensors are embedded in the servos and are used directly for posture adjustment.

The sensor implementation is based upon the above outlined general model: all sensors are composed from two distinct parts: cognitive and emotional, as shown in Figure C.3. The cognitive part is the function of the module, data processing, data transformation, etc. The emotional part is the meta setting for the cognitive part. It means that the emotion is not handling and modifying the user commands and robot sensory inputs with respect to its meaning, but rather emotions modifies the data only with respect to the body state. This can be seen in Figure C.3. In details, the emotional part affects the computational process from three different points of view. First, the emotional part simulates our intended idea that the information sensed from the environment is modified due to the emotional state of the hardware. Next, the process applied to the perceived input is parameterized by the emotional state and finally the output is post-processed and modified again by the emotional layer. Together, the emotional and the cognitive layer represent the general approach used all across the robot. This way all sensors and motors are inter-connected through the emotional layer and allow global changes based on the emotional state of the robot. This, as explained later, is done via the rewriting of the current robot instruction with respect to the robot's mood and emotional state.

C.4.1 Video-Sensor

Each feature of the vision sensor is running simultaneously and is not a part of a specific behavior. Rather, these processes are part of every behavior in a weighted manner. For example, the robot can be analyzing an object while

² the perceiving processes are vision processing, audio processing, etc.

Figure C.4: The Vision Sensor in this case has two input cameras, each assigned to one task as shown. Each operation on each signal is executed without the influence of the emotional component. Then the output of the transforms is altered by the emotion and finally a set of features are generated on the output.

having a discussion with a human or can be tracking the human eyes and performing a dance. In both cases, the behavior is not directly related to a single sensor thread or process but is emerging from the bottom-up modular and hierarchical approach.

The features of the vision system are affected by the emotion more than just on the behavioral level. Vision input starts with an image obtained from the camera (the leftmost arrow in Figure C.4). The emotions on this level modulate a set of general parameters (common for all vision processes). These parameters are: the image sampling rate F_s , the size of the image V_s and the number of processes N_p per signal. This can be seen on the left side of Figure C.4, in the emotional block. Once the features are obtained from the cognitive modules, emotions also modify the output on multiple levels. First the signal is integrated into the emotional state, and is registered by corresponding change of the energy. Next the signal is modified according to the history of the module activity and according to module-specific parameters. Finally the signal can be modified by the requirements of the current mood. These relations can be seen in the left part of Figure C.4.

The input control parameters represent the readiness of the visual sensor for input. For lower energy values the sensor will try to limit its usage by reducing the sampling rate of the image. This results in less energy consumption but in more inaccuracy (similarly in the case of the image size). The final general

parameter is used when the stereo operations are performed.

The output of the vision sensor is a set of features and/or commands. The rightmost part of Figure C.4 shows the various outputs that the vision sensor provides. Combined Noise Perturbations is the default signal. It is transmitted by default from the vision sensor when the vision sensor does not detect any feature. Valid input indicates whether or not it generated a valid output. Motion tracking signal provides either a set of coefficients representing detected movement or the set of commands that will allow the robot to track the moving object. The *face detection* signal carries the spatial and expression data about the detected face. Finally the *classified face/new face* is a signal representing the familiarity of the detected face.

At this stage of the development the main visual feature that the robot uses is the motion detection, while the face recognition/detection module is in the development. The whole video sensor is shown in Figure C.4.

The main feature threads of the signal are resumed in the Table below. As mentioned, the output of the vision sensor module are parameters and commands. From the robot actions point of view these features can be directly translated to simple higher level behaviors. The object tracking process can be directly parameterized for body-eye tracking so as the detected motion can be tracked by gravity. When active, this sensor feature will generate spontaneous robot motion and motion tracking. The object recognition and the face recognition have similar features in the sense that they both work on some particular objects. The object detection and its classification provides means to engage the robot's environment in order to make multi-sensory learning and representation for the robot behaviors. The robot can analyze environment and consequently engage in discussions with recognized elements from the environment, can use objects to insert topics into discussion or asks questions about particular objects. Finally, the face detection and recognition are the main factors in the initiation of the robot's communication and expression.

Video Sensor	
<i>Feature</i>	<i>Description</i>
Motion Detection	General motion detection, object tracking - spontaneous behavior generation
Object Detection	Object counting, recognition - spontaneous behavior generation (Incoming)
Face Recognition	Face recognition and memorization(Incoming) - spontaneous communication and expression behavior generation

Any of the visual features does not specify a particular behavior or emotional state. Rather, all features are running in parallel and are given resources based on their current utility to the current goal. For example during a conversation with a human, the robot uses face detection to recognize the speaker and to allow to remember possible previous experiences with this user. The motion detection is used to follow the speaker and also to allow evaluation of emotion based on the speaker motion. Finally the object detection is used as described above.

C.4.2 Audio-Sensor

Unlike the vision input system, the speech system is more limited in how much its performance can be affected by the emotional parameters due to the performance of the current technology. For example, it is desired to have the most accurate speech recognition possible, in order to allow easy human-robot verbal interaction. Thus there is no emotional influence in the speech recognition process and in the rhythm detection function. This can be seen on the left (input enters directly the cognitive module) of Figure C.5. The input to this sensor are two microphones, each used separately for different task or used in stereo tasks. Thus the general emotion parameter will include the number of tasks per input N_s . Once the input data is processed and before the output is generated the emotion affects this sensor in the same way as it does in the vision sensor module.

The output of the audio sensor generates the following features.

- Noise analysis is in principle similar to object detection, because it ex-

tracts sounds and patterns from the environmental noise. This is done by a thresholding algorithm working on a frequency spectrum of the recorded noise. Parameters such as the sampling frequency, the window of the FFT or the threshold are affected by the emotional system. This feature detects patterns such as footsteps sounds, door sounds, etc. Also stereo localization is part of this output. The output of these features are sequences of robot movements allowing to direct the robot towards the source of the detected object.

- Music parsing feature, takes music in mp3 format as input and breaks it down into a set of clusters, that represent direct mapping to servo motors. The emotional component in this module alters the magnitude of the motion, the desired position, and the speed and acceleration of the servo.
- The speech recognition, can be altered only in the way how it recognizes the words and sentences. It provides text of the recognized speech.

Figure C.5: The Audio Sensor has one or two microphones, each assigned to one task only. In this case with two microphones the robot runs only two tasks at the same time, the selection depends on robot's mode and behavior. The Music Analysis process can be used internally and does not require live music recording (music is played from recorded music files encoded in the mp3 format).

Again from the behavioral point of view, the audio sensor generates features that can be used either as simple behaviors or can be used for higher level behaviors. This is detailed in the table below. The audio sensor provides the interface for the user to communicate with the robot, to trigger particular

behaviors or to configure the robot. This is described more in details in Section C.6.

Audio Sensor	
<i>Feature</i>	<i>Description</i>
Noise Analysis	Reflex motion generation, orientation. Passive listening to the environmental noise and feature extraction (Incoming)
Music Parsing	Rhythm based behavior, list of commands directly usable.
Speech Recognition	Speech and communication: recognized speech can be directly used in dialog generator.

C.4.3 Voltage/Power-Sensor

Figure C.6: The Power Sensor has one single input, one per measured power line. The outputs are features indicating the energy/consumption measurement reflecting the energy evolution over time (differentials).

The main intero-ceptor is the Voltage/Power-meter connected to every servo in the robot and to every software module. This type of sensor measures only the robot internal state (in the form of power measure) and does not depend on the input. This allows not only to measure the consumed power by the robot but also allows the robot to find its most appropriate position (physical)

given its current physico-emotional state. In other words, the robot can self adjust the servo position in order to minimize the amount of energy that the servo consumes.

The hardware realization of this sensor can be seen in Figure C.6. This sensor has also its counter-part realized in software, realized in a similar way to previous sensors. The input are the coefficients indicating the power being used by the set of all servo motors.

Figure C.7: The Power-consumption Sensor.

The power sensor has a particular role in the robot beside sensing; it also determines the rate of evolution of the energy of the robot. Beside its hardware implementation as shown in Figure C.6 the power sensor is connected to every software module of the robot. It senses energy changes from every module and integrates them into the cumulative representation of the robot energy state. This is shown in Figure C.8. The energy representation is a set of virtual parameters that are coming from two sources: hardware and software. The hardware input is the measurement from all power sensors attached to all the motors of the robot. The software sensors are monitoring activity in the software modules.

It specifies the amount of energy that the robot is receiving. The amount of energy that the robot currently has will determine how its general state evolves. This is explained in more detail in Section C.7.

C.5 Actuators

The actuators, or the robot effectors are also of two types: software and hardware. The most common are servo controls but other devices can add con-

Figure C.8: The Power/Energy sensor - complete schema.

trols such as zoom, sharpness, voice pitch (for image and sound recognition controls), etc. Again an actuator is considered anything that can be send a command to execute and that can also understand a command in the internal robot language. Unlike the sensors, the actuator devices have only a flow of information inside themselves (Figure C.9).

Figure C.9: The generic actuator schema.

The implication of emotional influence in the motor is different from the one used in sensors. This is because the output of the actuator is a physical action and thus cannot be altered anymore using emotion. Thus the interaction between cognition and emotion on this level is limited to two levels. First, the command or the controlling sequence is modified by emotion as the expression of how well the servo is prepared to receive this command. Second, the cog-

nitive process within the actuator is modified in a similar way as it is done in the sensor.

C.5.1 Servo-Control

The control of the servos is resumed to the desired location, speed and acceleration. Every servo can be controlled individually or (as showed later) the groups of servos can be created in order to make more complex synchronized movements. The alphabet that the servo device accepts is specified by the CRL language.

Figure C.10: The generic actuator schema.

The robot has two necks and two heads (one head per neck). One neck has 8 motors and the other neck has 7 motors. The mechanical design of the robot has limitations in the extent of how far it can be moved. This is due to the way of how the servos work. For instance, because the servos have rotating axes, the force that they exercise has a non-linear distribution of the torque.

In this work, the focus is on the internal model of emotion with respect to simple robotic behavior and thus we do not provide any specific control mechanisms to achieve smooth robot motion or particular adjustments to the robot in order to make it look more socially and emotionally acceptable. Rather, the simple control of the robot is taken as the lowest level ³ of control and is designed to be used as building blocks to implement more complicated systems that will allow to express even more smooth behaviors.

Servo controller is one of the external hardware parts of the robot. All robotic heads use the maximum of 16 degrees of freedom, thus every head uses a single

³ the motor responses can be compared to unobservable individually controlled units that becomes socially interactive only when they are looked upon as a whole

ASC 16 controller. This board allows for commands controlling the servos in a parallel manner for their synchronous behavior.

C.5.2 Speech-Control

The speech generation is accessed using command `<speack>` or `<speech>`. Speech can be altered using the rate, volume and pitch. These settings are applicable to different voices. A schema of the speech control module is shown in Figure C.11. The cognitive part is implemented using FreeTTS (Text-to-speech software) that accepts text inputs and transforms them to the spoken language.

Figure C.11: The speech actuator schema.

The control of the speech and of the servos is the main tool for the expression of the behavior and emotions of the robot. Because Cynthea does not have face with a lot of degrees of freedom (only neck, jaw, eyes and voice are controlled) and because this robot does not have pre-defined emotions and behavior like in the Kismet or WE-4R robots [OS99, OS00], the emotional expression in my approach is completely dynamic. First, the emotion is constructed by the robot itself (there is no model per-se) and second, the model is based on functional basis, the emotion is dynamically created as the byproduct of the robot itself tuning its own activity. The robot does express its emotion through complex behavior that is dynamic over time. For example, the robot will express happiness by a certain set of neck and eyes motion and will express anger through motion and speech as well. Thus, the control of the motors and of the speech is highly sensitive to the context and to the desired behavioral response.

C.5.3 Network Control

The network interface is a hybrid device providing both input and output. Network interface is therefore a senso-motor device. It allows the robot to connect to the intranet so as to the Internet. The purpose of the network interface is to give the robot an almost infinite amount of information from web pages on the Internet. It allows the robot to seek for topics of discussion based on its moods. Also, the network interface allows the robot to be connected with other robots on the intranet network. It provides interface for robot-robot communication, behavior and action synchronization as well as a transparent interface allowing to observe every action that the robot does from the software-measurable parameters point of view.

The Internet interface by itself is represented as a command motor interface allowing the robot to contact a set of particular web sites allowing it to retrieve information about the weather, current news or to search for information by topic or by keywords.

The intranet interface is used during a group robot behavior or orchestrated planned behavior using a synchronizing server. In such a setting every robot knows what any other robot does. Therefore we avoid the problem of overloading the robot with complex sensory inputs in order to recognize and understand other robots behavior that are connected to the same network.

Figure C.12: The Network Senso-Motor Interface.

The network senso-actuator is influenced by the emotion in the following way. First, the output and input data that the robot generates and accepts from its environment, such as state information, command information, or processing information (Figure C.12), are generated or accepted based on this entire robot

emotional state. Thus the robot based on its emotional state will or will not accept other robots' data so as will or will not send its own private data.

The network interface has a particular role in the robotic theater build for this work. It is not only used to distribute data and commands, but it serves also as a transparent interface allowing the robots to see each other complete internal states. An illustration is shown in Figure C.13, where the representation of two networked robots is shown. As it can be seen, the networking interface has a similar function as the remaining robotic sensory inputs, because it serves to transmit commands, states and other information.

Figure C.13: Evaluation of a Humanoid Behavioral robot: The human-robotic interface (all observable behaviors and communications) and the Robot-Robot interface (Ethernet port where all states and robot activities are recorded)

C.5.4 Speech and Dialogue generation

The output of the speech recognition system is the text that is processed according to the mode of the recognizer. In the free user-robot interaction, the ALICE engine is used to generate the TTS based on the discussion and

the principle invented by Dr. Richard S. Wallace. Alice software is a pattern matching engine accepting XML-like type of scripting grammar (AIML - Artificial Intelligence Mark-up Language). It allows to represent the sentence in the language as a logical tree. The basic configuration allows ALICE to create BOTS, or virtual entities that have set of grammar and syntactic rules defining the dialogue generation. For more details refer to <http://www.alicebot.org/documentation/>. Here we provide a small example to describe the functionality of the Alice engine.

The AIML language understands recursive elements as well as text pointers that allows to match sentence patterns and generate answer. It can be described as a network of ordered IF-THEN rules. To store patterns in AIML the simplest way is to store a pattern in a one-to-one declaration (Code 1).

Code 1

```

<category> 1
  <pattern>ROBOT CAN YOU DANCE</pattern> 2
  <template> 1 3
    YES I CAN. 4
  <template> 1 5
</category> 6

```

Every rule in AIML is a *category* that matches a *pattern* to a *template*. The AIML code (Code 1), will be activated by the question *robot can you dance* and would generate *yes i can* . AIML has two wildcard operators *"_"* and *"*"* that can be used to match any sequence of string. Combined with the recursion operators *<srai>* and *<star>* more general rules can be created. The declarations (Code 2 - Code 4) show how to recognize *(very) good work robot* and *robot (very) good work* :

Code 2

```

<category> 1
  <pattern>_ ROBOT</pattern> 2
  <template> 1 3
    <srai> ROBOT <star index = "1"/></srai> 4
  <template> 1 5
</category> 6

```

This first code snippet recognizes any sequence /something/ robot and resends it as robot /something/.

Code 3

<code><category></code>	1
<code><pattern>ROBOT GOOD WORK</pattern></code>	2
<code><template> 1</code>	3
Thank you for the compliment.	4
<code><template> 1</code>	5
<code><category></code>	6

Code 4

<code><category></code>	1
<code><pattern>ROBOT * GOOD WORK</pattern></code>	2
<code><template> 1</code>	3
<code><srai> ROBOT GOOD WORK </srai></code>	4
<code><template> 1</code>	5
<code><category></code>	6

AIML allows to reduce language processing into simple rules and create general answers based on the input data.

The AIML bots are controllable through the input AIML scripts and can be changed in real time. This allows to use bots for different moods or topics. Cynthea right now uses a set of configuration files that represent speech for different moods. For example when Cynthea is in *good* mood, it will answer a question in a nice way, and when it is in "bad" mood it will answer with a less positive answer. However this approach is quite limited by the complexity of AIML itself. Thus further developments of this module are projected. Mainly a more general word/sequence generator based on the moods and on the internal state of the system is planned.

C.6 Control and Scripting

As described above, the robot is made from a group of modules that are organized in a functional hierarchy. This is represented in Figure C.14. There

are three types of modules with respect to the robot functionality: Sensors, Motors (or Senso-Motors) and Interconnect Modules. From Outside-In (from Sensors or Motors to higher behavior) the data flows from sensors through interconnect to motors. This is shown with the solid arrow in Figure C.14. In the bottom-up data flow, the information from the sensors is integrated to the Sensor Integrator Interconnect Module. On the other hand, the command flow (top-down) is starting in the Motor Integrator and decoder and continues down through the robot to the controlled devices (Figure C.14 dashed line). It can be seen by following the dashed lines, that the Sensor Integrator sends commands to the Motor Center, but not vice versa. This feature is due mainly to the fact that the actuators can be controlled but the information from the Motor center send to the sensor center is only informative; i.e. does not imply any response.

In other words, the sensor information can directly trigger a particular set of commands fed directly to the actuators, but the feedback from the motor center is reflected by a feedback from sensor to motor. On the top of Figure C.14 is the Interconnect Integrator. This is not a module in the strict sense of a single software or hardware entity, but it rather represents the various modules dedicated to higher level of control of the robot. For example, the Alice module used so far (soon to be replaced), the database interface or the quantum decision module (see Section C.7) are the Interconnect Integrator modules.

The command interfaces of the robot include the speech recognizer, the command line and the batch scripting. The speech recognizer is used for robot control, feedback and limited user interaction. The control mode of the recognizer allows the user to directly trigger various behaviors (interactive mode), call commands (direct mode) or configure (maintainer mode) the robot. The feedback mode is used for robot learning and robot self-adjustment and is described in details in Section C.3. Finally the recognizer can be used as a dialog manager or sentence recognizer to allow the spectator-robot interaction.

The robot is internally and externally controlled by a real time language, which describes robot's action at every moment. This is not a logic or predicate language but rather a real time action-flow descriptor ⁴ allowing the robot to execute a precise sequence of actions in space and time. The idea of CRL is

⁴ CRL specifies any input to every robot part at any moment of the time during the robot activity. CRL does not have logic elements or statements such as AIML has, but CRL allows to parse robotic activity to smallest elements of the CRL description language.

Figure C.14: Hierarchy of the control modules of the Cynthea Robot

to create a framework to control the actions and the cognitive activity of the robot while leaving freedom for the emotional expression. For example, the robot is asked to execute a task and according to its emotional state it will do it differently. This means that ideally the robot will simulate human behaviors in the context of the given task. If the task is to say 'Hello', the robot in normal mood will say Hello, the robot in bad mood might not say anything or might do some gesture expressing its bad mood. Similarly, a robot in a good mood might change Hello to something like 'Hello, what a beautiful day, don't you think so?'. To satisfy this, the language has the following properties:

1. it is hierarchical in order to capture the sequence and the priority of commands with respect to the robot architecture,
2. it is scalable in order to allow arbitrary architecture and complexity of the robot,
3. it is extensive in order to allow cooperation of multiple robots connected through network,

4. it is human readable in order to allow human operators to easily design scripts for various situations and purposes.

C.6.1 CRL - The language

Cynthea understands Common Robotic Language (CRL)⁵ scripting and commands created in year 2002-2005 by the author in the framework of this dissertation. CRL is an XML compatible scripting language with a purpose to completely describe robot cognitive behavior given a specific desired framework. It allows to completely specify the robot behavior by describing the behavior as a sequence of parameterized command tags. The commands that a robot can understand using CRL parser can be arbitrarily defined by using primitives of commands. These primitives are predefined for every device in the robot. This is described in more details later. CRL was designed in order to allow complete description of behavior for non-mobile (stationary) Humanoid robots. CRL language can be also extended to other types of behavioral robots.

The purpose of CRL is to describe a single or multiple behaviors/actions of a single robot or of multiple networked robots. First, CRL is 100% compliant with XML and the used parser is either a DOM or a SAX parser as described by the Sun Corporation. The description of each of the parsers can be found by visiting the <http://java.sun.com> and do a appropriate search or directly on the web site <http://www.saxproject.org/> for the SAX parser or <http://www.mozilla.org/projects/blackwood/dom/> for the DOM parser.

There already exists many languages for hardware, computer and other robotics related fields. The reason for creating CRL is mainly because we are focusing on features not described by most of the existing robot languages; features we are interested in are the behavioral, psychological and communication features for robots and devices having possibilities of emotional expression. CRL is powerful enough to describe any type of actions of the robot, and this is mainly because:

- It is designed to be arbitrary modifiable: this means no prescribed set of key words is defined beside the default package controlling the robot head.

⁵ CRL was developed at Portland Sate University by the Intelligent Robotics Group

- Its grammar is extensible, it can define as many rules as the complexity of the robot requires. This is achieved by over-writing and reusing the built-in commands of every robot module.
- It is multi-leveled; the language is suitable to describe both a single robot and all its behavior so as multiple robots and their timing, cooperation and coordination of individual activities.
- It is hierarchical, thus allows embedded commands within other commands, an example is to create more complex commands from smaller ones.

The idea of the CRL is, as already mentioned, to have a common description of robotics (smart robotics) devices designed for communication and interaction either with humans or with other robots via human-understandable interaction. The CRL is suitable for human reading and thus all tags are expressed in English. This allows in general to create files that describe single robots or groups of robots and their activity in time and space. This is done via two types of tags: direct and indirect. Direct tags describe temporal constraints and are shown below. Direct tags have for consequence time conditions. They help to organize on a time scale a complex set of behaviors or actions. In general, from the robot's point of view, the direct tags are commands to the robot treated as a single entity and thus they affect the whole robot. These tags are described later. The indirect tags describe directly an action or a behavior of a robot part. Indirect tags can be seen as local commands because they are specifically send to a robotic part such as an arm, mouth or eyes. The meaning of tags in CRL is as follows:

- Tags have no parameters and all variables are between tags. This means that for example if the robot wants to move left, then the sequence of commands will look something like this: `<move><left></left></move>`. In this case the example commands are direct because no destination was indicated.
- Each tag can represent either an action or an object.

Because we are using the JSAPI (Java Speech API) and JSML (Java Speech Markup Language) and it already has a parser each time the tag `<speech>` is encountered by the parser the tag is modified into `<jsmml>`. This is mainly done

in order to increase the understandability of the script that might get pretty long.

C.6.1.1 Command Description

The following commands are directly implemented in the CRL parser in order to provide a functioning robotic platform. As mentioned, the direct commands affect the whole robot but can be used to address single robotic part or element. Thus the main distinction when using direct or indirect command is (as already mentioned) the destination object.

- The Direct Commands are:
 - `<sync>` synchronize the actions and the destination object if they are children of this node
 - `<wait> #` pause the robot for a given amount of time (in seconds). Note, this command does not stop the robot instantaneously but rather waits until all actions in progress are finished.
 - `<flush>` clear all commands being executed and waiting for execution in all parts of the robot.

Direct commands allow also to give higher level orders to the robot. This includes specific behaviors (if available), activate higher level modules such as Dancing, Motion following and so on. However, in the case of higher level commands, the flow of the commands is not Top-down, but rather bottom-up-bottom. This means that the interface that first processes the command (the motor Integrator) passes the direct command upwards to higher level modules which are then sending commands back to the motor integration center or directly to the robot devices. Some of the higher level commands are shown below:

- The Direct Higher Level Commands are:
 - `<dance>` initiates the robot dancing behavior, it takes a song name as the parameter.
 - `<follow>` initiates robot to follow the motion detected on the camera.
 - `<mood>` sets robot mood.

- The Indirect Commands are
 - $\langle \text{move} \rangle \#$ move a given part to position defined in the range $[0,100]$. Note that this command can be multi-variable for the robot parts that use more than one servo-motor.
 - $\langle \text{close} \rangle$ close the given robot part; put the servo to position MIN
 - $\langle \text{open} \rangle$ open the given robot part; move the servo to position MAX
- The objects that can be addressed depend on the user definition of the robot, but some standard ones are provided here for the sake of clarity:
 - $\langle \text{cheek} \rangle$ each cheek can be accessed individually
 - $\langle \text{eyes} \rangle$ eyes move only together
 - $\langle \text{mouth} \rangle$
 - $\langle \text{eyebrow} \rangle$ individual access
 - $\langle \text{eyelid} \rangle$ individual access
- Specification for robot parts are:
 - $\langle \text{left} \rangle$ for individual access robot part only
 - $\langle \text{right} \rangle$ for individual access robot part only
 - $\langle \text{top} \rangle$ for individual access robot part only
 - $\langle \text{bottom} \rangle$ for individual access robot part only
- Finally some custom commands are available for specific robot parts:
 - $\langle \text{smile} \rangle \#$ parameterized smile $[0,100]$ (refer to the appendix section to check the size and number of parameters).
 - $\langle \text{frown} \rangle \#$ same as above
 - $\langle \text{speak} \rangle \#$ speak a sentence or a text with synchronized mouth movements; general tag indicating the sequence is a speech command. The quality of the synchronization between the spoken words and the jaw movements highly depends on the TTS synthesizer. Because Cynthia has a wooden face, with movable jaw, the synchronization is only on the level of the lip and jaw movement. More detailed synchronization is possible for robots that have artificial skin, movable lips and artificial tongue. All this information is provided with any standard TTS engine.

- `<blink>` blinks the eyes; note a random blinking is already defined as a automatic feature of the robot

The understanding of how this works is pretty easy. Each `#` indicates the parameter in a scale between 0 and 100 where 0 is defined as MIN and 100 is the MAX. Thus the command `<mouth><move>100</move></mouth>` results in the biggest possible opening of the mouth, also obtainable using the command `<mouth><open></open></mouth>`. When addressing individual elements such as cheek, eyebrow or so an additional identifier is required; every such an element has an identifier. Note that for an existing object such as cheek an invalid argument such as `<top>` will completely invalidate the command and the command will be not executed. Similarly if none of the tags is recognized the command will be discarded after a attempt to parse it is made by the robot. For example the command `<cheek><left><move>#</move></left><cheek>` results in moving the left cheek to position `#`. Of course this command can result in no movement because if the cheek is in its natural state, then the movement in the direction toward the natural relaxed state of the cheek will result in no movement. The order in which the CRL parser is working represents the hierarchy of the robot. The figure below explains the command robot structure.

Code 5

<code><crl></code>	<i>tag defining CRL language</i>
<code><robot></code>	<i>tag defining a robot device. All commands inside of this tag will be sent to one unique robot</i>
<code><face></code>	<i>tag defining a motor sequence of commands</i>
<code><smile> 50 </smile></code>	<i>tag defining a complex facial expression</i>
<code></face></code>	<i>end of commands to the face</i>
<code><face></code>	<i>tag defining a motor sequence of commands</i>
<code><normal> </normal></code>	<i>tag defining the resting facial expression</i>
<code></face></code>	<i>end of commands to the face</i>
<code><eyes></code>	<i>tag defining a motor sequence of commands</i>

```

    <move> 50 </move>
  </eyes>    end of commands to the eyes
  <mouth>    tag defining a motor sequence of com-
              mands
    <speech>
      Hello, this is a speech demonstration </speech>
  </mouth>    end of speech command
</robot>      end of script for this robot
<crl>         end of script file

```

The behavior defined by a CRL script is analyzed in similar top-down hierarchy to the picture. That is when a tag is met, the issued command by the parser is sent directly to the robot, which first checks if the tag is a direct command. If not the tag is analyzed in order to determine if it is a valid robot part. If yes the subsequent command is sent to the robot part which either executes it or not. The following example explains the idea.

When the robot is processing a CRL script, there is a set of rules that determines the manner of how the script is being processed. These rules are encoded in the CRL processor that every robot uses. First, every robot part can process only one command at a time⁶. This is because every command tends to use all resources of the given robot part, and thus running two commands on a single robotic part could result in a conflict or lock-up situation. Next, when a command is not recognized, the given part that could not process the command will discard it and will wait for the next one. The third rule is that when a command is given wrong arguments or parameters, the robot will either execute the given command with maximum or minimum values of arguments or will simply discard it.

From the above, it can be concluded that to synchronize commands the command *wait* should be used. In fact command *wait* will let terminate all previously started commands and will wait for a given period of time. While in a multi-robotic scheme, the sync command is used by the server computer to

⁶ Every robot can execute as many commands in parallel as it has different robot parts. For example a robotic head with N servos, and M higher-level modules, the robot will be able to execute M+N commands in parallel.

synchronize multiple robot actions. While in a single robot scheme the sync command is used in a different manner. Both approaches are illustrated below.

The script (Code 5) will execute all commands simultaneously as long as they do not belong to the same robot part. Thus, first, the face will smile, then it will return to normal. Then at the same time the eyes will move to the center position and the mouth will say the given text. The more robot parts connecting different devices are available, the more actions in parallel the robot can execute.

Code 6

```

<crl>
  <robot>
    <eyes>          commands for eye
      <blink></blink>
                        both eyes will blink
    </eyes>
    <face>
      <smile> 10 </smile>
    </face>
    <sync></sync>    processing of the script is interrupted
                        until no more commands are active,
                        in this case both commands 'blink'
                        and 'smile' must terminate.

    <mouth>
      <speech> My name is Cynthea </speech>
    </mouth>
    <wait>5</wait>
    <eyes>          commands for eye
      <blink></blink>
                        both eyes will blink
    </eyes>
    <mouth>
      <speech> and i am kinda happy </speech>
    </mouth>
  </robot>          end of script
<crl>              end of script file

```

Similarly to the Script 5 the Code 6 will synchronously execute the command *blink* in the eye robot part and the command *smile* in the face robot part. Then it will *wait* until all commands are done and only then it will *speak*, then *wait* and finally *speak* again. In the above example (Code 6), it can be seen that commands with parameters as well as commands without parameters are used.

Let's have a look at another example illustrating in more details the commands `<sync>` and `<wait>`. In the example (Code 7) the sync command is used to synchronize two robots connected through network so as their actions start at the same time.

Code 7

```

<crl>
  <sync>
    <robot> 1
      <eyes>
        <move> 50 </move>
      </eyes>
      <mouth>
        <open></open>
      </mouth>
    </robot>
    <robot>2
      <mouth>
        <speech>Text </speech>
      </mouth>
      <wait> 2 </wait>
      <eyelids>
        <close></close>
      </eyelids>
      <mouth>
        <speech>Text</speech>
      </mouth>
    </robot>
  <sync>

```

$\langle \text{crl} \rangle$

- The $\langle \text{sync} \rangle$ tags indicate, that the sequence of commands for both robots is synchronized in time.
- The $\langle \text{robot} \rangle 1$ tag indicates that the commands will be issued for the robot #1.
- The $\langle \text{robot} \rangle 2$ tag indicates that the commands will be issued for the robot #2.

The Script 7 illustrates other properties of CRL. First, from examples 5, 6 and 7 it is obvious that no indirect tags can be mixed so as no incoherent commands ⁷ can be created. Each time an action or a sequence of speech commands is introduced in the script the sequence must be finished before another sequence is started. To synchronize these actions the keyword $\langle \text{sync} \rangle$ or $\langle \text{wait} \rangle$ can be used. Second, there is a strict order of tags. This means that a CRL script can be a sequence of commands for a single robot, a synchronized sequence of commands for a single or multiple robot parts or a synchronized script for multiple robots as shown in Code 7. Other side features apparent are: the movements are parameterized on a scale between 0 and 100; a eye movement with value of 50 will bring the eye in the middle while 0 and 100 will be extremities of its movement radius. As the steps of the parameters are of size 1 on an interval between 1 to 100, the wait command parameters have steps of size $\frac{1}{100}$ of a second; we assume that for the technology (used for this project) these time steps are precise enough with regards to the hardware and software speed.

C.6.1.2 Robot Constructor

A sub-language of CRL is used to describe the current robot and construct a software model of the controller. The constructor file specifies the following parameters of the robot:

⁷ Commands that the CRL parser will define as invalid or unknown are considered incoherent. This can result either from syntactic error or from command error.

1. The robot parts and its devices. Also it allows to specify the ranges of the servo motors, their initial speed and acceleration.
2. The commands, the initial set of commands as well as the custom commands can be specified
3. Commands for robots and also the commands for their parts can be specified

Figure C.15: The Command Hierarchy structure.

The following constructor (Code 8) builds a robot head with controllable mouth and eyes. First the *robot* and the first robot-part command *mouth* is created in lines 2 – 3. By default the robot is created only as a shell for higher level or global commands that are described for each functionality separately. Moreover most of the high level features are embedded by default or their function is directly higher-level-command-dependent. From the previous discussion, it can be seen that the robot is designed as a modular architecture⁸. Thus the higher level commands are either global commands such as shown in Figure C.15, or specific modules such as dancing, tracking, reacting, etc. The main control through CRL is used on the robot part level, and also most of the commands are on this level (Figure C.15) when designing behavioral robots.

⁸ The robot is a set of modules, every modules being connected in a hierarchical tree.

Code 8

```

<crl>
  <robot> robot
    <part> mouth
      <devices>0 1
        <min>100 0</min>
        <max>100 10</max>
      </devices>
      <command>smilebroad
        <targets>move ignore 70</targets>
        <targets>move ignore 60</targets>
        <targets>move ignore 70</targets>
        <targets>move ignore 60</targets>
      </command>
    </part>
  </robot>
</crl>

```

The Code 8 shows the robot shown in Figure C.16 and it defines one new command: *smilebroad*, that is in fact four small movements of the servo motor. Different commands can be designed in a similar way. This shows how to define subroutines in CRL in order to complexify the language and simplify its usage by the user. The subroutines have no parameters: this is due to the fact that because each subroutine can be based on an arbitrary number of subcommands, there is not a simple way how to parameterize the subroutine in a general way. For example, *smilebroad*, has four consecutive commands, and thus parameterizing macros will be directly related to the number of commands used in this macro. Rather, a sequence of subroutines can produce desired behavior of the robot, depending on the parameterization of each of them. For example subroutines can be defined to explicitly execute some type of motion such as the above *smile broad* command. Other subroutines can be defined in order to set robot in specific configuration by setting motor properties such as speed, acceleration and others.

As can be seen, one feature of CRL is the command rewriting. For the explanation of this let us assume the following example (The pseudo code 8 above). A command 'smilebroad' was defined in the mouth object and was parameterized according to some specifications (number of degree of freedom,

Figure C.16: Example of tree command structure for a simple robot.

expression, etc.). Now if someone wants to have different smiles but does not want to create a new command for the simplicity of the script it is possible to parameterize the 'move' command by writing it as a normal movement command so as $\langle move \rangle \# \langle /move \rangle$ becomes $\langle smile \rangle \# \langle /smile \rangle$. In the case the number of parameters is correct the smile command will be executed with the given parameters. In the other case where the parameters do not satisfy the definition of command smile, only parameters fitting the definition will be used and the invalid ones will be discarded.

C.6.2 Scripting

To begin with the robot commands, there are three main classes in the CRL: RobotPart, RobotDevice and RobotCommand. The syntax of the language combined with the three main classes of the robot components represents then directly the different levels of the processes in the robot. The structure of the

script (Code 9) represents the robot hierarchy:

$$\begin{aligned}
 &\langle \text{crl} \rangle \\
 &\quad \langle \text{robot} \rangle \\
 &\quad \quad \langle \text{RobotPart} | \text{RobotCommand} \rangle \\
 &\quad \quad \quad \langle \text{RobotDevice} | \text{RobotCommand} \rangle \\
 &\quad \quad \quad \quad \langle \text{RobotCommand} \rangle \\
 &\quad \quad \quad \quad \quad \langle / \text{RobotCommand} \rangle \\
 &\quad \quad \quad \quad \langle / \text{RobotDevice} | \text{RobotCommand} \rangle \\
 &\quad \quad \quad \quad \langle / \text{RobotPart} | \text{RobotCommand} \rangle \\
 &\quad \quad \langle / \text{robot} \rangle \\
 &\quad \langle / \text{crl} \rangle
 \end{aligned} \tag{C.1}$$

CRL thus distinguishes between two main types of commands: Direct commands and Indirect commands. Indirect commands are those that are affecting the robot as a whole; in general they are used to set a robot to a particular state, to synchronize robot actions or to start a particular behavioral module (such as dancing, visual tracking, etc.). The indirect commands are affecting the robot from the Top-Down specification point of view. This can be seen in Figure C.18. It can be noticed that the whole script can be considered as an indirect command, because from the general point of view it forges the behavior from the top-down side. For example equation C.2 is a command that sets the robot in the *happy mood*.

$$\langle \text{robot} \rangle \langle \text{mood} \rangle \text{happy} \langle / \text{mood} \rangle \langle / \text{robot} \rangle \tag{C.2}$$

Indirect commands, on the other hand are all instructions directly manipulating robot devices such as servos and speech, or those that are modifying parameters of some devices. On this level of CRL, the devices can use custom commands such as

$$\langle \text{head} \rangle \langle \text{move} \rangle 14 \ 98 \ 34 \langle / \text{move} \rangle \langle / \text{head} \rangle \tag{C.3}$$

The parameters between the inner tags are directly robot implementation dependent. As all parameters are in the range $[0; 100]$, the values represent the relative position in the specified range. Thus in this case, the three servos in

Figure C.17: Servo motor position and ranges. Top row shows a servo at position 0 and 100, while the three bottom ones together are the schematic representation of the command $\langle head \rangle \langle move \rangle 14 \ 98 \ 34 \langle /move \rangle \langle /head \rangle$

the robot-part head will move respectively to $\approx 15\%$ of the range of the first servo, $\approx 100\%$ of the range of the second servo and finally to $\approx 35\%$ of the range of the third servo motor (Figure C.17).

The definition and functionality of the CRL allows to specify a complete and deterministic framework for the robot actions and behavior. The framework is the main keystone in this approach to study dynamic emotions.

C.7 Artificial Emotions

So far, the emotional model that is used in this robot has not been discussed. The reason is that because the proposed model is a constructivist approach (not directly based on knowledge of human or animal emotions), to explain the emotion as a whole, the complete robot architecture and structure must be known.

The emotions used in the CYNTHIA robot are represented on the energy modules of every part of the robot creating a pool of emotional agents. Each of the agent represents points in the emotional state based on the parameters of the energy modules. Parameters of the energy are robot dependent. Every robot created in the framework presented here, will have emotional parameters based on their particular hardware-software representation. Each software-hardware part of the robot is based on a set of predefined functional parameters, and all modules are sharing the same set of all possible param-

Figure C.18: Two main classes of commands also represent the structure of the robot. The direct commands go straight to processing by the devices, while the indirect commands require either command preprocessing or the integration of external processes.

eters to be used. Because all of the emotional agents have the same number of parameters, they can be represented in a unique common emotional space. In this space, modules are using only emotional parameters required for the module functionality. These parameters are the primitives of the emotional state space. These states are updated by individual energy functions of every module. One meta agent ("robot-agent") collects the data from all other ones and determines the general state for the robot as a whole. The measure of energy of this meta agent determines also the robot's "*mood*". The role of these emotional agents is to coordinate their energy income and consumption in order to approach the optimal value by altering the emotional parameters. All together, these agents represent the robot body in the sense that every robot code has its representation.

The energy of the robot is the evaluation of the all agents by the meta agent, while the energy of every agent is based on a competition principle. Every emotional cell consumes some energy and makes prediction about its future expenses. The meta agent, collects the energy of every cell and determines the maximum of allowable energy for every cell based on the global-local op-

Figure C.19: The complete robot schema - with only main components displayed

timum evaluation. Thus the amount of energy for every cell is proportional to the agent's prediction on consumption if available through the robot-agent. Because the amount of information in the robot varies, the pool of emotional agents will be changing the number of elements.

The embodiment of the emotion is represented in two aspects: the emotional modules are not self-existent (do not have other function than to monitor existing robot parts or processes) and the emotional module is a part of a pool with other emotional modules. The emotional model is constructed based on the basis given from the "robot body itself". In each robot, the emotional model varies, because it is based on the level of the customization of the robot. The individual emotional model for each module is shown in Figure C.20. There are seven basic human-like emotions implemented: anger, depression, normal, melancholy, happiness, fear and contentness used in our robot. Four of them are aligned in the energy plane and they are (anger, depression, melancholy and happiness). Fear and Contentness are related to how well the input matches robot expectations. The circles in the Figure C.20 represent level lines of a Gaussian energy distribution such as in Figure C.21. The emotional agent searches the emotional space defined by these references with respect to the optimization of its energy state.

Thus, for example, the robot receives a command C. Being in body-emotional state E, every emotional agent related to the command process will search for the best solution and will modify the command C. The execution of the command will modify the energy of the cell, and of the whole robot. The meta agent evaluates the current robot emotional state with its goal achievement and it modifies the energy allocation appropriately. The allocation rule of the agents follows these two rules:

- Agent activity is appropriate to its current energy balance
- Agent activity is desired by the robot

These two rules have for consequence the constant redesign of the hierarchy by the agents. In the first case, the amount of the energy that the agent requires from the meta agent is completely covered by the amount obtained. This means that such a software agent will move up and down based solely on its own need. In the second case, the agent will be required to obtain a certain amount of energy and thus will be adapting the function to the given energy. Each agent computes how much energy it needs to remain as close as

possible to the ideal value of energy. As each agent wants to be close to its optimal energy value, agent forced to spend more energy will try to spend it while the agents that lack energy will try to preserve their own stock. This means that there will be exploration on every level of behavior; whether an agent is satisfied or not.

In the case of the language the modifications of the energy are the changes in the verbal and motor activity of the robot. For example, the robot is processing the CRL command $\langle head \rangle \langle move \rangle 13 \ 23 \langle /move \rangle \langle /head \rangle$. This command can be changed in three different ways.

1. command can be *expanded* to multiple commands by local modifications. This is the case when an agent obtains more energy than it requires. Also, command expansion is a way of generating commands to a an inappropriate current emotional state. For example, if a given device is not processing any command and its energy increases, this device will generate behavior in order to approach the energy optimum.
2. command can be *shortened*. This is the inverse of command expansion. In this case, commands can be skipped or combined. For example successive movement to different positions can be replaced by a single movement to the final position.
3. command can be *modified*. The possibilities of executing this command depend on the parameters of the functional module. Parameters such as speed, acceleration, rate or volume can be used to modify the command.

Using these transforms, each agent alters its energy evaluation and moves in the emotional space. An example of emotional space model is shown in Figure C.20. The x-axis corresponds to the action space and thus it is translated to various commands via the energy calculations. The y-axis represents the two emotions, fear and contentness, that depend on inputs. First, when a given input is processed, the fear/contentness assessment is calculated. The definition of fear in the present context is: the ratio between the perception and the readiness of the robot to react. Thus the less the robot is able to follow a given scene the more its fear will increase. This will result in depression and a behavior where robot will wait until it will be able to intervene again. Similarly if a robot is presented with exactly the same information it will transfer to melancholy and the robot will present behavior that will allow it to explore

Table C.4: Emotional Relation

	Energy			
$\frac{\partial E}{\partial t}$	Trend	$\langle Opt. \rangle$	$\asymp 0$	$\rangle Opt.$
+	Content	Melancholy	Happy/Normal	Depression
+	Fear	Melanch./Depress.	Aggress./Normal	Depress./Melanch.
-	Content	Depress./Melanch.	Happy/Normal	Melanch./Depress.
-	Fear	Depress./Melanch.	Aggress./Normal	Melanch./Depress.

new contexts. As both fear and contentness are defined as gradients, the size of the emotional impact on the command can be controlled. Table C.4 illustrates this principle.

The four states placed around the energy level lines in Figure C.20 are symmetric through the center. In fact, every agent can very quickly change from happy mood to angry mood, without any external input. The difference between these two moods is based on agent history. Every agent tracks its own fear/contentness ration, or how well was it performing in its past. The actions that contribute to moving a cell from optimal energy value are: increasing the fear, and (opposite) increasing the contentness.

Figure C.20: Emotional mapping in the action space. The circles, are the energy level map. Emotional States can be represented as energy states modulated by the two dimensional Fear/Contentness.

The four emotional states were placed so as to represent the desired behavior.

- Happiness - robot is open to communication with appropriate behavioral scheme corresponding to a robot presenting "happy" behavior identified by the user. *Command expansion and modification.*
- Melancholy - robot is still open to communication but its behavioral scheme does not support its thoughts. *Command modified.*
- Anger - robot is closed to communication with appropriate behavioral scheme corresponding to a robot presenting "angry" behavior identified by the user. *Command shortened and modified .*
- Depression - robot is closed to communication but its behavioral scheme does not support its thoughts. *Command modified.*

All four emotions coexistent at the same time in every agent and the strongest has the most important impact on the general behavior of the robot. Other more subtle emotional states such as joy, soothness, unhappy, tired or disgusted can be expressed based on the Figure C.20. However, because the focus is not to explicitly build a emotional model (but rather build a framework allowing the robot to evolve and express emotions), only the main categories are used here to explained the initial hierarchy of emotional states.

In robotics, the emotions are in general based on biological models such as [Dam99, Dam04, Ekm93, Ekm03], while in this dissertation the emotions are directly mapped to the robot states, state transitions and the general system time-evolution. Here the emotions are defined upon the equilibrium of the robot with respect to the current environment. This is translated to the fact that *a)* there are emotions that are related only to the internal state, *b)* there are emotions related to the input and *c)* the general state is the result of the interaction of the two previous categories of emotion. For example the (A,V,S) model implemented in Kismet is represented in a simpler way in our approach and is also more realistic with respect to the generality of emotions.

In general, the energy function looks much like the one shown in Figure C.21 and as already mentioned represents the body state. The modification of the commands and the agent processing, is called Strategy. Unlike in the emotional space, the Strategy space is purely defined in the space of possible modification to command and to the process it self. The strategy is represented by a quadratic or a multi-quadratic function as shown in Figure C.22. The strategy represents the actions that the emotion can do to the body. Both

Figure C.21: The energy function is similar to the Gaussian Distribution curve.

functions are explained later in details, however it is important to remark that the strategy will tend to go against the trend of the energy, in order to be as close to 0 as possible. Similarly to energy, here the strategy evaluation function represents the difference between the current and the optimal value. In this case the optimal value is also 1; if a given process matches the situation to a one-to-one mapping, the optimal strategy was obtained.

The emotional model defined above is implemented in the Cynthea robot as Vector based control over the functional parameters of the robot. These parameters represents the basic control elements and can but have not to be all used by a given functional module. Table C.5 shows the basic elements of the Cynthea emotional state.

The emotional element associated with every functional module of the robot is based on a common premise of specifying states and transitions from states using self-adaptation and learning.

Table C.5: Emotional Parameters for control

Device	Parameters	Other
Camera	Brightness, Colors, etc	Affects higher level transforms
Microphone	Pitch, Sampling, etc	As above
Temperature	Range, Rate, Resolution	
Gyroscope	Precision	
Touch Sensors	Sensitivity	
Servo	Speed, acceleration, position	
Energy	Precision	Primary - hardware safety

Figure C.22: The strategy function is a multi-quadratic parameterizable function.