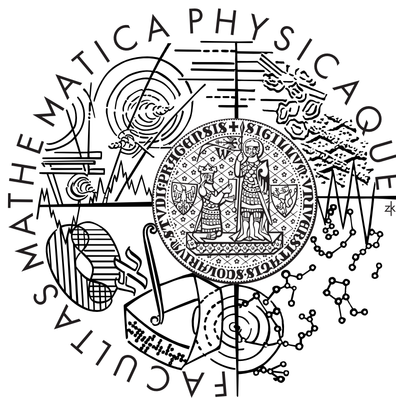
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Robert Špalek

Space Complexity of Quantum Computation



Katedra teoretické informatiky

Vedoucí diplomové práce: RNDr. Pavel Pudlák, DrSc.
Studijní program: informatika

Praha 2002

Poděkování

Na tomto místě bych v první řadě rád poděkoval vedoucímu mé diplomové práce RNDr. Pavlu Pudlákovi, DrSc. za veškerou pomoc a čas, který mi věnoval. Díky němu jsem se blíže seznámil s kvantovými výpočty a vypracoval tuto diplomovou práci. Rovněž mu děkuji za velice užitečné připomínky týkající se zpracování mé diplomové práce a za mnohé diskuze nad teoretickými problémy.

Dále bych rád poděkoval Prof. dr. Harry Buhrmanovi a Dipl.-inform. Hein Röhrigovi z Centrum voor Wiskunde en Informatica v Amsterdamu za přečtení textu a užitečné připomínky a Michael van der Gulikovi z Vrije Universiteit v Amsterdamu za jazykové korektury.

Můj dík si zaslouží také D. E. Knuth za vynikající typografický systém TEX, Bram Moolenaar za textový editor VIM, Otfried Schwarzkopf za grafický editor IPE a všichni lidé podílející se na projektu GNU/Linux.

V neposlední řadě bych chtěl poděkovat Áji za její trpělivost, všem svým přátelům a především svým rodičům za všestrannou podporu, kterou mi během studia poskytovali.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Amsterdamu dne 16. dubna 2002

Robert Špalek

Contents

1 Preliminaries **1**
- 1.1 Syllabus . . . 1
- 1.2 Contribution of the thesis . . . 2
- 1.3 Notation . . . 3

2 Branching Programs **5**
- 2.1 Deterministic Branching Programs . . . 5
- 2.2 Nonuniform and uniform sequences of BP's . . . 7
- 2.3 Probabilistic Branching Programs . . . 10
- 2.4 Complexity classes . . . 13
- 2.5 Reversible computation . . . 15

3 Quantum Computation **19**
- 3.1 Intrinsics of Quantum Computation . . . 19
  - 3.1.1 State space . . . 19
  - 3.1.2 Evolution . . . 21
  - 3.1.3 Measurement . . . 23
- 3.2 Quantum Circuits . . . 25
- 3.3 Quantum Turing Machines . . . 26
  - 3.3.1 Definition . . . 27
  - 3.3.2 Transition function . . . 29
  - 3.3.3 Quantum behaviour and language acceptance . . . 29
  - 3.3.4 Complexity measures and complexity classes . . . 30
  - 3.3.5 Nonuniform version . . . 32
  - 3.3.6 Special forms of QTM's . . . 33

4 Quantum Networks **35**
- 4.1 Raw Quantum Networks . . . 35
  - 4.1.1 Definition . . . 35
  - 4.1.2 Consistency of the model . . . 38
  - 4.1.3 Uniform sequences of QN's . . . 44

4.1.4 Language acceptance and complexity measures . . . 48
4.2 Special forms of QN's 50
4.2.1 Layered QN's 50
4.2.2 Oblivious QN's 51
4.2.3 Bounded degree QN's 52

5 Equivalence of QN's and QTM's 53
5.1 Converting a QTM to a sequence of QN's 53
5.2 Converting a sequence of QN's to a QTM 58

6 Converting QN's to special forms 63
6.1 Layered layout 63
6.2 Oblivious layout 67
6.3 Bounded degree layout 70
6.4 Connected graphs 76
6.5 Conclusion 78

7 Achieving reversibility 79
7.1 The back-tracking method 79
7.2 The infinite iteration method 88
7.3 The Pebble game method 95
7.4 Tradeoffs between the methods 101

8 Space bounded Quantum Computation 105
8.1 Recognising NC_1 languages by 5-PBP's 105
8.2 NC_1 is contained in 2-EqQBP 107

List of Tables

2.1 Acceptance modes of PBP's 12

4.1 Transition functions μ_0, μ_1 of the QN in Figure 4.2 41

7.1 Computation of the PBP in Figure 7.8 99

List of Figures

2.1 Example of a BP computing x_1 & x_2 & x_3. 7
2.2 The lack of the parental condition in a BP 16
2.3 Two trivial examples of RBP's computing the sum $x_1 + x_2 + \ldots + x_n$ modulo 2 and 3 16
2.4 A nontrivial example of a RBP computing x_1 & x_2 17

3.1 Bloch sphere representation of a qbit 20
3.2 Relation between the fanout and parity gates 27
3.3 Design of a Quantum Turing Machine 28

4.1 Structure of a raw QN 41
4.2 Example of a QN 42
4.3 A well-formed QBP not fulfilling the parental condition . . . 43
4.4 The interference pattern of a QBP 44
4.5 Example of a layered QN 52

5.1 Structure of the monochromatic graph of a QTM 55

6.1 Constructing a layered QBP 65
6.2 Converting a layered QN to an oblivious QN 68
6.3 Representation of an independent set of 3 rotations by a QBP 71
6.4 Representation of a 6×6 unitary operator (that needs not be permuted) by a QBP 74
6.5 Reachable vertices in the QBP of the Hadamard operation . 76

7.1 Sketch of the back-tracking method 80
7.2 Fulfilling of the parental condition in a BP 81
7.3 Inconsistent cyclic path obtained using the back-tracking method 83
7.4 Adding dummy vertices to a real BP 84
7.5 Performing the back-tracking on a real BP 84
7.6 Gaining acceptance probabilities of a QBP 91
7.7 Scheme of the Pebble game 95

7.8 A PBP converted using Pebble game 99
7.9 Tradeoff between the Pebble game and back-tracking 102

Title:	Space Complexity of Quantum Computation
Author:	Robert Špalek
Author's e-mail:	`robert@ucw.cz`
Department:	Katedra teoretické informatiky
Supervisor:	RNDr. Pavel Pudlák, DrSc., MÚ AV
Supervisor's e-mail:	`pudlak@matsrv.math.cas.cz`

Abstract: A new quantum model of computation — a sequence of Quantum Branching Programs — is proposed and its consistence is verified. Its power, properties, special forms and relations to existing computational models are investigated. Both nonuniform and uniform variants are considered. It is shown that QBP's are equivalent to Quantum Turing Machines and that they are at least as powerful as their deterministic and probabilistic variant. Every QBP can be converted to its most regular form (layered, oblivious, 2-bounded degree, with connected graph) at low cost. Almost all these simulations preserve the space complexity and enlarge the time complexity at most polynomially, moreover we prove that the target sequence obtained by converting a uniform source sequence is also uniform — we describe an explicit construction machine for each conversion. It is demonstrated that width-2 oblivious QBP's recognise NC_1 languages in polynomial time.

Keywords: quantum branching programs

Název práce:	Prostorová složitost kvantových výpočtů
Autor:	Robert Špalek
E-mail autora:	`robert@ucw.cz`
Katedra:	Katedra teoretické informatiky
Vedoucí dipl. práce:	RNDr. Pavel Pudlák, DrSc., MÚ AV
E-mail vedoucího:	`pudlak@matsrv.math.cas.cz`

Abstrakt: Navrhli jsme kvantový výpočetní model — posloupnost kvantových branching programů (QBP) — a ověřili jeho vnitřní konzistenci. Vyšetřili jsme jeho schopnosti, vlastnosti, speciální formy a vztahy k zavedeným výpočetním modelům. Uvážili jsme jeho neuniformní i uniformní variantu. Ukázali jsme, že QBP jsou ekvivaletní s kvantovými Turingovými stroji a že jsou alespoň tak silné jako jejich deterministická nebo pravděpodobnostní varianta. Každý QBP může být převeden do svého základního tvaru (vrstevnatý, zapomnětlivý, stupně vrcholů maximálně 2, propojený graf) s malou ztrátou. Téměř všechny předložené simulace zachovávají prostorovou složitost a prodlužují časovou složitost nejvýše polynomiálně, navíc je dokázáno, že konverzí uniformní posloupnosti vznikne také uniformní posloupnost — uvádíme explicitní konstrukční stroj pro každou konverzi. Ukázali jsme, že zapomnětlivé QBP se šířkou 2 jsou schopny rozeznat NC_1 jazyky v polynomiálním čase.

Klíčová slova: kvantové branching programy

Chapter 1

Preliminaries

1.1 Syllabus

We propose new[1] quantum models of computation — Quantum Networks and Quantum Branching Programs. We investigate their power, properties, special forms, and relations to existing computational models. The document has the following structure:

Chapters 2 and 3 are introductory chapters. In Chapter 2, we remind the computational model of deterministic Branching Programs and formulate the definitions in our notation. We define the probabilistic and reversible variants and uniform sequences of BP's. In Chapter 3, we introduce the notions of quantum computing. A very short recapitulation of quantum matters (state space, evolution, measurement) is included, though this document does not serve as an introduction course of quantum computing. We also mention Quantum Circuits and define properly Quantum Turing Machines and their nonuniform version.

Chapters 4–7 comprise the research topic. Chapter 4 proposes the desired models — Quantum Networks and their acyclic variant Quantum Branching Programs. We study various aspects of this concept and give several examples. We define uniform sequences of QN's, complexity measures, and a language acceptance. The chapter is concluded by definitions of various special forms of QN's.

Chapter 5 proves the equivalence of QN's and QTM's in both nonuniform and uniform case. Chapter 6 investigates special forms of QN's — the layered and oblivious layout, bounded degree quantum operations and using connected graphs. It concludes that every QN and QBP can be converted into the most regular form at little cost.

[1] during writing of this document, the models of QBP's have already been published

Chapter 7 investigates the relation between quantum, deterministic and probabilistic BP's, i.e. the simulation of the latter ones by QBP's. The major obstacle of the conversion is the reversibility of quantum computations. We show that for both BP's and PBP's, there are two methods achieving reversibility — the first one is space preserving and time consuming, the second one uses an additional space, but the time overhead is not so big. We also present time-space tradeoffs between them. We show that the acceptance probabilities of a PBP are squared by the conversion.

Chapter 8 is not a part of the research. It just deals with a topic related to the space complexity of quantum computations. It shows that one quantum bit is enough to recognise any NC_1 language in polynomial time.

The electronic version of this document can be found on Internet at URL `http://www.ucw.cz/~robert/qbp/`, you can also contact the author by e-mail `robert@ucw.cz`.

1.2 Contribution of the thesis

The presented model of Quantum Branching Programs was greatly inspired by Quantum Turing Machines described in [Wat98]. Though similar models have been described in [AGK01, AMP02, NHK00, SS01] during writing of the document, the results presented here are original. This text also contains slightly more detailed description than the referred articles.

In Section 4.1.2, we define an important concept — a so-called *family decomposition* of a transition graph. Having defined it, a proof of the inner consistency of the model is straightforward and moreover several simulations follow directly from it.

We have deliberately designed the uniformness of sequences of Quantum Networks in such a way, that the obtained model is equivalent with Quantum Turing Machines. The simulation of a QN by a QTM is straightforward thanks to using families at intermediate stages of the simulation. The reverse simulation is simpler, because the model of QN's is less structured in principle.

We have designed special forms with respect to improving the physical feasibility of the computational model. The layered layout reduces the quantum space complexity of the program. The oblivious layout decreases the number of quantum operations that need to be prepared. However, due to the generality of the model, the conversion procedures are not very efficient for a general QN. Nevertheless several algorithms can be efficiently expressed directly in the oblivious layout.

We have properly scrutinised the bounded degree layout and found an optimal conversion procedure. On the other hand, all existing implementations of quantum computers operate on qbits nowadays, hence the decomposition of an operator into 2-bounded degree quantum branches is not directly useful for them.

All methods achieving reversibility discussed in Chapter 7 have already been published before. However we have adjusted them for QBP's. The time-space reversibility tradeoffs presented here are original results, though the deterministic tradeoff has already been published in [Ben89].

1.3 Notation

Let us mention the notation used in this thesis. As usually, $\mathbf{N}$, $\mathbf{N}_0$, $\mathbf{Z}$, $\mathbf{Z}_n$, $\mathbf{R}$, $\mathbf{R}_0^+$, and $\mathbf{C}$ will denote natural numbers, natural numbers including 0, integers, integer numbers $\{0,1,2,\ldots,n-1\}$, real numbers, non-negative real numbers and complex numbers.

For any set S, both $\#S$ and $|S|$ will denote the number of elements in S and $P(S)$ will denote the potency of S, i.e. the set of all subsets of S. For any set S, S^n is the set of all sequences of S of length n, S^* is the set of all finite sequences of S. For any mapping f and set S, $f(S)$ will denote $\{f(s)|s \in S\}$.

Let $f_1, f_2 : \mathbf{N} \to \mathbf{R}_0^+$ be functions mapping natural numbers into nonnegative real numbers. We say that

- $f_1 = \mathrm{O}(f_2)$ iff $\limsup_{n\to\infty} f_1(n)/f_2(n) < \infty$,
- $f_1 = \mathrm{o}(f_2)$ iff $\lim_{n\to\infty} f_1(n)/f_2(n) = 0$,
- $f_1 = \Omega(f_2)$ iff $\liminf_{n\to\infty} f_1(n)/f_2(n) > 0$,
- $f_1 = \Theta(f_2)$ iff $f_1 = \mathrm{O}(f_2)$ & $f_1 = \Omega(f_2)$.

For any finite or countable set S, $\ell_2(S)$ will denote the Hilbert space whose elements are mappings from S to C. Elements of such spaces will be expressed using Dirac notation; for each $s \in S$, $|s\rangle$ denotes the elementary unit vector taking value 1 at s and 0 elsewhere, and $\{|s\rangle | s \in S\}$ is the set of base vectors of $\ell_2(S)$. For $|\phi\rangle \in \ell_2(S)$, $\langle\phi|$ denotes the linear functional mapping each $|\psi\rangle \in \ell_2(S)$ to the inner product $\langle\phi|\psi\rangle$ (conjugate-linear in the first coordinate rather than the second). The norm of the vector is defined in usual way as $\| \, |\phi\rangle \, \| = \sqrt{\langle\phi|\phi\rangle}$.

For a matrix M, M^T denotes the transpose of the matrix, M^* denotes the complex conjugate of the matrix and $M^\dagger = (M^T)^*$ denotes the adjoint of the matrix. We say that a matrix $M : \ell_2(S_1) \to \ell_2(S_2)$ is a ***block matrix*** iff

it is composed from smaller blocks, the blocks are another matrices. We say that M is a ***block-diagonal*** matrix iff the blocks are centred around the diagonal and all blocks that do not intersect the diagonal are null.

For any oriented graph $G = (V, E)$, $d_G^-(v)$ and $d_G^+(v)$ will denote the input and output degree of vertex v (i.e. $d_G^-(v) = \#\{w \in V | (w, v) \in E\}$). If it is obvious which graph we are talking about, the subscript G may be omitted. We say that a vertex v is a ***source*** (a ***sink***, an ***internal vertex*** respectively) iff $d^-(v) = 0$ ($d^+(v) = 0$, $d^+(v) \neq 0$ respectively). If $(v, w) \in E$, then we say that v is a ***parent*** of w and w is a ***child*** of v. We say that a graph is ***acyclic*** iff it does not contain an oriented cycle. We say that a graph is ***connected*** iff there exists an undirected path from v to w for any two vertices v, w. The ***component*** of a graph is the maximal set of vertices connected to each other.

We say that a mapping $f : \mathbf{N} \to \mathbf{N}$ is ***time constructible*** (***space constructible*** respect.) iff there exists a Turing Machine M with time complexity (space complexity respect.) f.

If we do not use a parameter in a definition, a theorem, or a note, we often do not waste a letter for it, but substitute an '_' character instead. For example $P = (_, _, _, Q, E, q_0, _, _)$.

Chapter 2

Branching Programs

A Branching Program is a model of computation that represents a decision process based on the values of input variables. It is represented by a graph whose vertices correspond to the internal states of the Branching Program. Every vertex is labelled by a number of the input variable queried in this state, every outgoing edge is labelled by a letter from input alphabet.

The Branching Program model was developed with respect to obtain the least structured computational model. Lower bounds proved for this model obviously hold for the other models as well, since they are easily reduced to this one.

Branching Programs need not have a regular program structure. The program behaviour can vary from step to step and the internal memory represented by current state needs not be accessed in bits. Therefore the programs can be very optimised for a particular problem.

Despite the generality, lots of interesting lower bounds and tradeoffs have been proved, see [Bea89, CS83, MNT90].

2.1 Deterministic Branching Programs

We concentrate ourselves on the decision Branching Programs that produce no output except for the label of the destination state. There exists also a model in which the programs can produce an output during the computation. It is implemented by a second labelling of the edges – every edge contains a list of output bits produced when passing via this edge.

Branching Programs were discussed in [Sip97, Weg87, Weg00], but we give our own definition to fit the notation to our purposes.

Definition 2.1 A *Branching Program* (shortcut BP) is an ordered sequence $P = (n, \Sigma, \Pi, Q, E, q_0, d, v)$ such that

- $n \in \mathbf{N}$ is the length of the input,
- Σ is the input alphabet,
- Π is the output alphabet,
- Q is a set of graph vertices,
- $E \subseteq Q^2$ is a set of oriented edges,
- $q_0 \in Q$ is a starting vertex,
- $d : Q \to \mathbf{Z}_n \cup \Pi$ is a function assigning input variables to internal graph vertices and output result to graph sinks,
- $v : E \to \mathcal{P}(\Sigma)$ is a function assigning the letters of input alphabet to the graph edges,

and the following requirements hold:

i. (Q, E) is an acyclic oriented graph,

ii. $d(q) \in \Pi$ iff q is a sink of the graph,

iii. for every internal vertex and input alphabet letter there is exactly one edge starting in the vertex and labelled by the letter, i.e.

$$(\forall q \in Q, d(q) \notin \Pi)\ (\forall \sigma \in \Sigma)\ (\exists! e \in E) e = (q, _)\ \&\ \sigma \in v(e).$$

Note 2.1 When not explicitly given, we assume the input and output alphabets are taken $\Sigma = \Pi = \{0, 1\}$ and the input size n is chosen automatically according to the labels of internal vertices.

The computation of a BP proceeds as follows. It starts in the starting vertex. While the current vertex is not a sink, the decision based on the value of appropriate input variable is made and the internal state is changed to the destination of appropriate outgoing edge. When the computation arrives to a sink, the label of the destination vertex is produced at the output. The result of a BP P on input x will be called $P(x)$.

A very simple example of a BP computing the logical AND of three input variables ensues in Figure 2.1. Henceforth, if not stated else, the solid edges will be labelled by 0 and the dashed ones will be labelled by 1.

In the following text we will discuss languages recognised by BP's. A language L is a set of strings $L \subseteq \Sigma^*$.

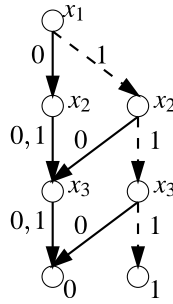


Figure 2.1: Example of a BP computing x_1 & x_2 & x_3.

Definition 2.2 We say that a BP P for input size n ***recognises*** (or ***accepts***) the language $L \subseteq \Sigma^n$, if

$$(\forall x \in \Sigma^n)\ x \in L \Longleftrightarrow P(x) = 1.$$

The complexity of BP's is measured in terms of the length of the longest computational path and the program size.

Definition 2.3 Let P be a Branching Program. We define that the ***size*** of P is the number s of its vertices, the ***space complexity*** of P is $\lceil \log_2 s \rceil$ and the ***time complexity***[1] of P is the length of the longest computational path, i.e. the longest oriented path in its graph going from the starting vertex.

2.2 Nonuniform and uniform sequences of BP's

A Branching Program is capable to solve a problem of a fixed input size n. To overcome this boundary we need to introduce sequences of BP's.

Definition 2.4 A ***sequence of Branching Programs*** P is an infinite sequence $P = \{P_i\}_{i=1}^{\infty}$ of BP's where for every input size a special program is provided.

Definition 2.5 We say that a sequence of BP's $P = \{P_i\}_{i=1}^{\infty}$ ***recognises*** the language $L \subseteq \Sigma^*$, if

$$(\forall x \in \Sigma^*)\ x \in L \Longleftrightarrow P_{|x|}(x) = 1.$$

[1]we shall talk mostly about the maximal time in this thesis, for defining the expected time a probabilistic distribution on the inputs must be chosen

Definition 2.6 Let $t, s \in \mathbf{N} \to \mathbf{N}$ be a time constructible (space constructible respect.) function. We say that a sequence of BP's $P = \{P_i\}_{i=1}^{\infty}$ has the time complexity t (space complexity s respect.) iff, for every input size n, P_n has the time complexity $t(n)$ (space complexity $s(n)$).

If we do not place other restrictions on the sequence, the model is incredibly powerful, because it contains all languages including the non-recursive ones. For every input size n and language $L_n \subseteq \Sigma^n$ there exists a particular BP recognising that language, hence there also exists a sequence of those BP's recognising any fixed language $L = \bigcup_{k \in \mathbf{N}} L_k$.

This phenomenon can be circumvented by a notion of uniform sequences. The problem sketched in the previous paragraph is caused by the fact, that such sequence of BP's just exists and we have no clue how to find it. We can rid of those sequences by requiring the uniformness condition. This condition says that the BP's must be constructed by an effective algorithm.

Every BP can be readily encoded by a binary sequence of 0's and 1's. We can imagine that the vertices are identified by distinct identifiers and listed one after one in the ascending order including their labels. Then a sorted list of edges ensues, every edge lists its source and destination vertex and their labels. It is straightforward that every BP is (not necessarily uniquely) encoded by some sequence. The sequence length in records is asymptotically equal to the size of the BP up to a multiplication constant, since the output degree of the vertices is bounded and thus there are asymptotically no more edges than vertices.

Let us fix a particular encoding of BP's.

Definition 2.7 A sequence of BP's $P = \{P_i\}_{i=1}^{\infty}$ is called *uniform* if there exists a Turing Machine M that for given $k, b \in \mathbf{N}$ yields the b-th bit of the encoding of P_k. Such machine M is called a *construction machine* of P. If there is no such machine, we call the sequence *nonuniform*.

The Turing Machine model of computation is not described here, since it is widely known. It is defined, for example, in [Gru97, Pap94, Sip97]. A Quantum Turing Machine model is described in Section 3.3.

It turns out that the Turing Machine and Branching Program models of computation are equally strong both in the nonuniform and uniform case. Moreover if the complexity of the construction machine is bounded, these models also turn out to be equivalent in the sense of time and space complexity.

The simulation of a Turing Machine by a Branching Program is quite straightforward:

Claim 2.1 Every Turing Machine M with advice that stops in finite time for every input can be simulated by a sequence of Branching Programs P in the same time and space. If the machine M uses no advice, then the resulting sequence P is uniform.

Proof. The proof will be just outlined: for every input size n the vertices of P_n will correspond to the configurations of M and the edges of P_n will accord to the transition function of M. The time and space complexities will obviously be preserved. If M uses no advice, its activity is regular and completely described by its (finite) transition function. Hence the structure of P_n is also regular and the sequence P can be generated by a Turing Machine M' depending only on M. □

Notice that for every language $L_n \subseteq \Sigma^n$ there exists a BP P_n recognising L_n with size at most exponential in n. It needs not be longer, since if P_n forms a balanced tree having $|\Sigma|^n$ leaves, it can be adjusted to every language L_n by just tuning the output result in every leave. However for some languages substantially smaller programs can be constructed.

Hence if we allow the length of the advice of Turing Machines to be exponential in the problem size then the opposite simulation is also possible. Let us consider that asking for advice by Turing Machine is implemented by executing a special instruction after the inquiry is written on a special tape. This topic is discussed in detail in Section 3.3.5.

Claim 2.2 Every sequence of Branching Programs P can be simulated by a Turing machine with advice M in the same space and polynomially (in input size) longer time. The advice length is asymptotically equal to the size of the corresponding BP, i.e. it is not longer than exponential in input size.

Proof. Sketch of the proof: The advice of M will contain a complete description of P_n in some fixed encoding, thus the advice depends only on the input size as needed.

M has a work tape describing the current state s of P_n and another work tape containing the intermediate results. Every step begins by asking the advice for the description of behaviour of P_n in the state s. There is a little impediment in doing this, because we do not known exactly, where in the advice the description of s is stored. Since the lists of vertices and edges are sorted, we can find the index by binary search, that can be done in polynomial time.

When M knows the variable P_n decides on, it shifts the input tape, reads the symbol and chooses a proper destination state which is then copied

to the first work tape. The loop is executed until the advice claims the computation has finished.

The maximal length of second work tape is linear in the length of first work tape, thus M works in the same space. The computation of every step of P_n by M takes time at most polynomial in n (polynomial time for the binary search in the advice, linear time for shifting across the whole input tape), hence the time overhead is only polynomial in n. □

Claim 2.3 Every uniform sequence of Branching Programs P constructed by a Turing Machine M' can be simulated by a Turing Machine M. The space complexity of M is the sum of the space complexities of P and M'. The time complexity of M is the product of the time complexities of P and M' and a polynomial in n.

Proof. Sketch of the proof: The simulation proceeds in a similar way to the one in the proof of the previous claim. Instead of asking for advice the construction Turing Machine M' is launched to produce the desired bits of the description. This causes the claim on the additional time and space. □

2.3 Probabilistic Branching Programs

It is well known that computational models become much more powerful when the randomness is taken into account. Many probabilistic algorithms that beat their deterministic counterparts are known nowadays. These algorithms typically base their decisions on random numbers and their results are erroneous with some small probability.

Like in the previous section, we shall define a probabilistic version of a BP, describe its computation and define the language acceptance of PBP's.

Definition 2.8 A *Probabilistic Branching Program* (shortcut PBP) is an ordered sequence $P = (n, \Sigma, \Pi, Q, E, q_0, d, p)$ such that

- $n, \Sigma, \Pi, Q, E, q_0, d$ have the same meaning as they had in Definition 2.1 of a BP on page 5,
- $p : E \times \Sigma \to \mathbf{R}_0^+$ is a transition function assigning probability to the transition through an edge given a letter from the input alphabet,

and the following requirements hold:

i. (Q, E) is an acyclic oriented graph,

ii. $d(q) \in \Pi$ iff q is a sink of the graph,

iii. the transition probabilities of distinct outgoing edges with a fixed label sum to 1, i.e. let $E_q = \{e \in E | e = (q, _)\}$, then

$$(\forall q \in Q, d(q) \notin \Pi)\ (\forall \sigma \in \Sigma)\ \sum_{e \in E_q} p(e, \sigma) = 1.$$

The computation of a PBP proceeds similarly to the computation of a BP. The only change is that after we extract the value of the appropriate input variable, the new internal state is chosen randomly according to the transition probabilities.

For a given PBP P, an input word $x \in \Sigma^n$ and an output result $\pi \in \Pi$, we define $p_\pi^P(x)$ as the probability of obtaining result π when the PBP P is run on the word x. The superscript P may be omitted when it is clear which P is meant from the context. The same notation is used also for sequences of PBP's.

Note 2.2 The space and time complexities are defined in a similar way as the complexities of a BP in Definition 2.3. The space complexity defined for a BP is suitable also for a PBP, but the time complexity needs a little adjustment. If we want to be more accurate, we should define not only the ***maximal time complexity***, but rather the ***expected time complexity***, since the computation and its length can depend on random choices. If $p_{\pi,k}(x)$ is the probability that the computation of a PBP P on input x stops in time k yielding result π, then $p_\pi(x) = \sum_{k \in \mathbf{N}_0} p_{\pi,k}(x)$ and we denote the expected time complexity of P on x by

$$\text{exp.time}^P(x) = \sum_{k \in \mathbf{N}_0} \sum_{\pi \in \Pi} k \cdot p_{\pi,k}(x).$$

Further, the time complexity of P can be taken as the maximum of time complexities over all inputs $x \in \Sigma^n$ or, after a probabilistic distribution on the inputs is chosen, the expected value can be computed.

The language acceptance can be defined in many ways using distinct criteria, each of them is suitable for other tasks and leads to other complexity classes.

Definition 2.9 Let P be a sequence of PBP's, $L \subseteq \Sigma^*$ be a language, m be a mode[2] listed in Table 2.1. We say that P ***accepts*** L ***in mode*** m iff for every

[2]the mode names are carefully chosen to accord the common complexity class names, e.g. NP, BPP, co-RP, ...

m	$x \in L$	$x \notin L$	comment
Eq	$=1$	$=0$	never makes mistake
R	$\geq 1/2$	$=0$	bounded one-side error
co-R	$=1$	$\leq 1/2$	bounded one-side error
N	>0	$=0$	unbounded one-side error
co-N	$=1$	<1	unbounded one-side error
B	$\geq 3/4$	$\leq 1/4$	bounded error
Pr	$>1/2$	$\leq 1/2$	unbounded error

Table 2.1: Acceptance modes of PBP's

$x \in \Sigma^*$ the probability $p_1^P(x)$ of obtaining result 1 fulfils the appropriate condition listed in the table (there are 2 constraints there depending on whether or not $x \in L$).

Note 2.3 The bounds $1/2$, $3/4$, $1/4$ in modes R and B are not magical, setting 2δ, $1/2+\delta$, $1/2-\delta$ would serve as well for any fixed $\delta \in (0, 1/2)$. The error probability can be decreased below any fixed bound by repeating the computation and choosing the logical OR of results (in the mode R) or the majority (in the mode B). The proof is straightforward in the first case, since if consecutive computations are independent, the probabilities of yielding erroneous results multiply. In the second case, the proof involves a simple application of the Chebychev's inequality for probability distributions. Even if we allow δ_n to be polynomially small in the problem size n, i.e. $\delta_n = \Omega(1/p(n))$, the probabilities can still be improved by this method.

On the contrary, if the error probability in modes N and Pr is exponentially close to 1, i.e. $\delta_n = O(e^{-n})$, it is quite infeasible to yield a correct solution by the algorithm.

Note 2.4 The modes R and co-R (N and co-N respect.) are counterparts, because if a language L is recognised in mode m, then its complement co-$L = \Sigma^* - L$ is recognised in mode co-m.

The co-modes of the modes Eq, B and Pr are not present in the table, because they are equal to the original modes.

Note 2.5 If we do not restrict the vertex output degree, it could happen that the state of a computation spreads too fast into distinct vertices, e.g. a random choice from $k = \Omega(n)$ adjacent vertices can be done in unit time

by setting the probability $1/k$ to each of the k outgoing edges. This causes problems when simulating a PBP by a probabilistic TM, because a TM has only a fixed number of internal states for all input sizes. The power of PBP's is not decreased if the number of distinct random choice types is finite, since a complex random choice can always be simulated by a chain of simple random choices within accuracy exponential in the length of the chain.

Definition 2.10 For a vertex $q \in Q$ and an input letter $\sigma \in \Sigma$ the *random choice type* of vertex q and letter σ is the sorted sequence[3] of probabilities of the edges labelled by σ outgoing from q:

$$\text{type}(q,\sigma) = \{p(e,\sigma) \,|\, e = (q,_) \in E \;\&\; p(e,\sigma) \neq 0\}.$$

We say, that a sequence of PBP's has a *finite number of random choice types*, if the set of random choice types for all input sizes is finite, i.e.

$$\left| \bigcup_{\sigma\in\Sigma} \bigcup_{n\in\mathbf{N}} \bigcup_{q\in Q_n} \{\text{type}(q,\sigma)\} \right| < \infty.$$

Note 2.6 It turns out that it suffices if there is just one random choice type allowed, e.g. a fair coin flip $\{\frac{1}{2},\frac{1}{2}\}$. However it is more convenient to introduce a special random choice type in some cases (e.g. $\{\frac{1}{3},\frac{1}{3},\frac{1}{3}\}$, instead of simulating it inaccurately every time.

2.4 Complexity classes

Having described the criteria of languages acceptances by PBP's and the complexity measures of PBP's, we shall now divide languages into complexity classes. Every class will contain languages acceptable by PBP's using somehow bounded resources.

Definition 2.11 We define that $m-\text{TIME}(t(n))$ ($m-\text{SPACE}(s(n))$ respect.) is the class of languages recognised by uniform sequences of PBP's with time complexity $t(n)$ (space complexity $s(n)$ respect.) in mode m. The uniform sequences must be constructed by a Turing Machine M' running in

[3]an item may occur more times in the sorted sequence

polynomial time and polylogarithmic space (in input size).[4]

$$
\begin{aligned}
m\text{–TIME}\,(t(n)) = \; & \{L \subseteq \Sigma^* \mid (\exists \text{uniform sequence of PBP's } P_L) \\
& P_L \text{ has time complexity } \mathrm{O}\,(t(n))\ \& \\
& P_L \text{ accepts } L \text{ in mode } m\}, \\
m\text{–SPACE}\,(s(n)) = \; & \{L \subseteq \Sigma^* \mid (\exists \text{uniform sequence of PBP's } P_L) \\
& P_L \text{ has space complexity } \mathrm{O}\,(s(n))\ \& \\
& P_L \text{ accepts } L \text{ in mode } m\}.
\end{aligned}
$$

Note 2.7 We have already shown in Claims 2.1–2.3 that uniform BP's and TM's simulate each other with low time and space overhead. The same can be stated for their probabilistic counterparts. One direction is straightforward, but the simulation of a PBP by a PTM has little impediments. The proof is simple when the number of random choice types is finite, then the same simulation method can be used by reserving a special decision state for every random choice type. If the set is not finite, then we must implement the simulation of a complex random choice by a chain of simple ones in addition. It complicates the program but it does not change anything else. Hence the complexity classes just defined correspond approximately (up to the overhead of the simulation) to the commonly used ones. In particular, if we define

$$
\begin{aligned}
m - \text{POLYTIME} &= \bigcup_{k \in N_0} m\text{–TIME}\left(n^k\right), \\
m - \text{POLYLOGSPACE} &= \bigcup_{k \in N_0} m\text{–SPACE}\left(\log^k n\right),
\end{aligned}
$$

we see that thanks to the constraints on the construction TM M' of the uniform sequence of PBP's these complexity classes are equal to the common ones.

The common complexity classes are well described in every textbook of computational complexity, e.g. in [Gru97, Pap94, Sip97].

Note 2.8 The languages accepted in mode Eq have been defined in terms of PBP's. We shall show that nothing changes if we replace the PBP by a BP in this case. The proof is very simple: the PBP P never makes mistake, hence each its computational path with nonzero probability gives correct answer at the sink. Therefore it does not matter which particular path is chosen. If we modify the transition probabilities of P in the way that the leftmost path (for any definition of leftmost) is chosen in every vertex, we get an equivalent deterministic BP P'.

[4] i.e. $M' \in SC$ — Steve's class

2.5 Reversible computation

In this thesis, we shall investigate a quantum version of branching programs. One of the intrinsics of the quantum world is that every process is reversible. To fit a program into quantum computer, not only its interface but also every its computational step must be locally reversible.

It is not immediately apparent that this limitation does not matter so much, i.e. that every classical program can be converted into its reversible variant at little cost. We shall show in Chapter 7 that there are indeed more methods doing that, each of them having its own advantages and drawbacks.

Let us define how a reversible BP looks like and examine a few examples.

Definition 2.12 A *Reversible Branching Program* (shortcut RBP) is an ordered sequence $P = (n, \Sigma, \Pi, Q, E, q_0, d, v)$ such that $n, \Sigma, \Pi, Q, E, q_0, d, v$ have the same meaning as in Definition 2.1 of a BP on page 5 and the following requirements hold:

i. (Q, E) is an acyclic oriented graph,

ii. $d(q) \in \Pi$ iff q is a sink of the graph,

iii. the parental condition must be fulfilled — for every vertex q there is exactly one common input variable assigned to all parents of q:

$$(\exists d_p, d_p : Q \to \mathbf{Z}_n)\ (\forall q_p, q \in Q) \quad (q_p, q) \in E \Rightarrow d(q_p) = d_p(q),$$

iv. in both ingoing and outgoing direction, every vertex has either no adjacent edges or exactly one adjacent edge for every letter from input alphabet

$$(\forall q \in Q, d_G^-(q) > 0)\ (\forall \sigma \in \Sigma)\ (\exists! e \in E) \quad e = (_, q)\ \&\ \sigma \in v(e),$$
$$(\forall q \in Q, d_G^+(q) > 0)\ (\forall \sigma \in \Sigma)\ (\exists! e \in E) \quad e = (q, _)\ \&\ \sigma \in v(e).$$

Note 2.9 The parental condition may look somehow odd at a first glance. However it is essential if we want the reverse direction of computation to be unique, like the forward step is. We decide on exactly one input variable in the forward step, hence the same we expect in the opposite direction. There is a simple counterexample shown in Figure 2.2 of a reversible-like BP that fulfils all other requirements but is not reversible at all — when $x_1 = 0\ \&\ x_2 = 1$, the reverse step is not unique.

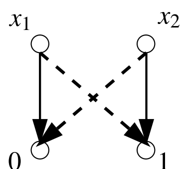


Figure 2.2: The lack of the parental condition in a BP

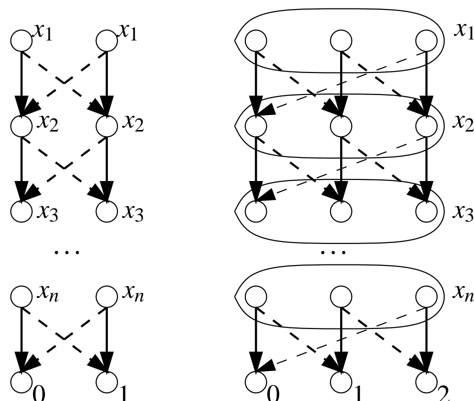


Figure 2.3: Two trivial examples of RBP's computing the sum $x_1+x_2+\ldots+x_n$ modulo 2 and 3

Two trivial examples of well formed RBP's are shown in Figure 2.3. They compute the parity (sum of the bits modulo 2) and the sum of the bits modulo 3. They both have a very regular structure, in Chapter 4 we would say that they are layered and oblivious. Recall that the solid lines are labelled by 0 and the dashed lines are labelled by 1.

A program in Figure 2.4 computing the logical conjunction of two variables x_1 & x_2 serves as a nontrivial example of a RBP. The starting vertex is the left upper one. It has not so regular structure, in fact it is a pruned version of a RBP generated from a (non-reversible) BP computing the same function by the back-tracking method achieving reversibility. This and some other methods will be developed in Chapter 7. Notice that the number of sources equals the number of sinks and that there is no consistent[5] path going from the starting vertex to the sink Error.

[5]a path is called consistent if there exists an assignment of input variables, for which it is a computational path

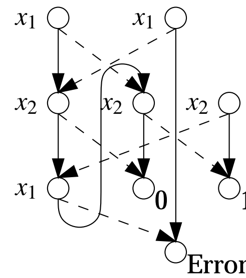


Figure 2.4: A nontrivial example of a RBP computing x_1 & x_2

Note 2.10 As seen in Figure 2.4, the requirement we indeed require is the local reversibility — the computation must be reversible at every instant for every setting of input variables. Even if the programmer knows for sure that if the program state has arrived to a particular vertex, then the input variables must have fulfilled a particular equation, he must design the program to be reversible at every instant for every setting of input variables.

Note 2.11 It is apparent that unless the graph of a RBP is just a single path, it must have more sources than one. They will never be reached during the computation, but they are important for fulfilling the reversibility requirements.

Chapter 3

Quantum Computation

We shall not supply an introduction course of Quantum Computation here. Many splendid books have been published about it, e.g. [NC00, Pre99]. However for the purposes of the completeness of this document, we shall remind the basic principles and the notation in this chapter.

3.1 Intrinsics of Quantum Computation

A classical computer is situated in a unique and observable state at every instant of the computation. It is possible to dump its memory into a storage medium, examine or modify it and restart the computation from the interrupted state any times we want. The computation is also deterministic and unless an error occurs, it always yields the same result.

A quantum computer behaves completely else, which we shall remind in this section.

3.1.1 State space

A quantum computer can be situated in a superposition of classical states. Its state is completely described by a ***state vector*** $|\varphi\rangle \in \ell_2(S)$ of complex amplitudes,[1] where S is the set of classical states, e.g. for an n-qbit computer $S = \{0,1\}^n$ thus $|\varphi\rangle \in \mathbf{C}^{2^n}$. The quantum analogue of a bit is called a ***qbit***.

The set of ***classical states*** of an n-qbit quantum computer is called a ***computational basis*** and the states are labelled by $|i\rangle, i \in \{0,1,\ldots,2^n-1\}$. We say that the linear combination $\sum_i \alpha_i|\varphi_i\rangle$ is the ***superposition*** of states $|\varphi_i\rangle$ with the ***amplitude*** α_i of the state $|\varphi_i\rangle$.

[1] we shall ignore the notion of mixed states in this thesis, i.e. all states considered will be the pure states

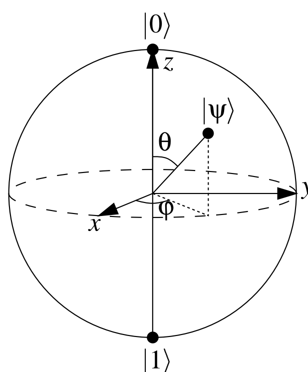


Figure 3.1: Bloch sphere representation of a qbit

Note 3.1 A state vector must fulfil the ***normalisation condition***, which is a quantum analogue of the requirement that the probabilities of distinct events sum to 1. It says that $|\varphi\rangle$ is a unit vector, i.e. $\langle\varphi|\varphi\rangle = 1$. It should also be reminded, that the ***global phase*** is unobservable for computational purposes, hence we do not distinguish between $|\varphi\rangle$ and $\alpha|\varphi\rangle, |\alpha| = 1$.

Example 3.1 An one qbit computer has two classical states $|0\rangle$ and $|1\rangle$ and it can also be situated in the superposition of these states, e.g. $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ or say $\frac{|0\rangle-2i|1\rangle}{\sqrt{5}}$.

Note 3.2 There is a visual way of representing a qbit. A qbit may be situated in a general superposition state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Since $\| \, |\psi\rangle \, \| = 1$, $|\alpha|^2 + |\beta|^2 = 1$ and we can rewrite the equation as

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right),$$

where θ, φ, γ are real numbers. We shall ignore the unobservable global phase $e^{i\gamma}$, hence the qbit state is completely described by two real variables. The numbers θ, φ define a point on the unit three-dimensional sphere, often called ***Bloch sphere***, see Figure 3.1. It provides a useful visualisation of most one qbit quantum operations. Unfortunately there is no direct generalisation to multiple qbits.

Note 3.3 It turns out that the ***phase*** of an individual quantum state in a superposition state is a quantity of the same importance as the identity of

the state itself. This phenomenon is distinct from the fact, that the global phase is unobservable. We can not distinguish between $|0\rangle + |1\rangle$ and $-|0\rangle - |1\rangle$ while distinguishing between $|0\rangle + |1\rangle$ and $|0\rangle - |1\rangle$ is trivial thanks to the Hadamard operation defined in the next subsection.

Note 3.4 Notice that the state space of a joint system is a tensor product of the state spaces of the individual systems. The composite state $|\varphi\rangle|\chi\rangle$ is also denoted by $|\varphi\chi\rangle$.

Nevertheless it is not true that every composite state is a tensor product of the individual states. This phenomenon is called *entanglement* and such individual states are called entangled. There is an extremely important example called EPR-pair which serves as a useful tool for many applications (super-dense coding, quantum state teleportation and many others; see [NC00]). One of the four EPR-pairs can be written as

$$|\chi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}},$$

the other three differ by the sign or by flipping the first qbit.

3.1.2 Evolution

A computational step of a quantum computer is described by an *evolution operator*. Quantum physics requires that this operator must be *unitary*, i.e. reversible and norm-preserving. If a quantum computer is situated in the state $|\varphi\rangle$, then it switches to the state $U|\varphi\rangle$ after the operator U is performed.[2] Recall that the product of unitary operators is also a unitary operator.

Example 3.2 Every permutation operator is unitary, hence any classical reversible operation is also permitted in the quantum world. The simplest operators on a qbit are perhaps the *identity* I and the *bit flip* operator X. The operators have the following representations in the computational basis $|0\rangle, |1\rangle$:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The proper quantum examples of evolution operators are the *phase flip* operator Z and the combined flip operator Y. The operators I, X, Y, Z form

[2] the quantum evolution is indeed a continuous process described by a Hamiltonian, but this fact is not important for our purposes

a basis over the space of one qbit operators, they are called *Pauli operators* and they will be mentioned a few times in the document.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The last but not the least important operator is the *Hadamard operator* H.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

The Hadamard operator has a very interesting property. If we apply it to the basis state $|0\rangle$, we obtain $(|0\rangle + |1\rangle)/\sqrt{2}$, which is a *uniform superposition* of either states. If we apply it once more, we obtain the basis state $|0\rangle$ again.

Note 3.5 It is very illuminating to imagine how do the one qbit operators act on the Bloch sphere. The identity I leaves the state unchanged. The Pauli operators X, Y, Z reflect the state through the x, y, z axes respectively. The Hadamard operation H performs a rotation of the sphere about the y axis by 90° followed by a reflection through the $x - y$ plane.

Another interesting examples are the *rotation operators*. A simple algebra shows, that the operator

$$R_x(\theta) = e^{-i\theta X/2} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} X$$

rotates the Bloch sphere about the x axis by angle θ. Similar operators $R_y(\theta)$, $R_z(\theta)$ can be defined also for other axes.

Example 3.3 The *controlled NOT* operator (shortcut CNOT) acts on two qbits. It has the following representation in the computational basis $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

If the first qbit is nonzero it does nothing, otherwise it flips the second qbit.

Note 3.6 Quantum physics allows us to apply *any* unitary operation in unit time, at least in principle.[3] As a matter of fact, only a small set of

[3] e.g. the operator SAT that solves the satisfiability problem and flips the output bit if the input problem has a solution is a unitary operator

operations is physically feasible. It can be shown (see [NC00]) that there exists a small (discrete) set of operations that forms a *universal set*, i.e. every quantum operation operating on two qbits can be simulated by a finite sequence of the universal operators with the precision exponential in the length of the sequence. Moreover every quantum operation on n qbits can be decomposed into a finite sequence of two qbit operations. This sequence is unfortunately exponentially long for most operators — it can be proved by a simple counting argument.

The goal of the quantum computational complexity is finding which operators can be implemented more efficiently and describing such implementations. This is what the quantum algorithm design is about.

3.1.3 Measurement

From a computer scientists point of view quantum physics provides a powerful tool for a fast multiplication of exponentially large matrices of complex numbers. Though the matrices need to be unitary and easily decomposed, it seems to lead to an exponential speedup over classical computers. However we encounter a substantial problem at the moment — the amplitudes of a quantum state are hidden and protected from direct observations.

The only way how to obtain information from a quantum computer is observing it. The *observation* is described by an *observable*. Every observation inevitably disturbs the quantum state and projects the state vector into some vector subspace. The more information we get by the measurement, the more we disturb the state.

Definition 3.1 An *observable* is a collection $\{M_m\}_m$ of measurement operators. The index m refers to the measurement outcomes that may occur in the experiment. If a quantum computer is situated in the state $|\varphi\rangle$ immediately before the experiment, then the probability that result m occurs is $p(m) = \| M_m|\varphi\rangle \|^2$ and the state of the system after the measurement is

$$\frac{1}{\| M_m|\varphi\rangle \|} M_m|\varphi\rangle.$$

The measurement operators must satisfy the *completeness condition*

$$\sum_m M_m^\dagger M_m = I.$$

Note 3.7 The definition of the observable just stated is very general and allows us to perform so-called *POVM measurements* (see [NC00]). For our

purposes, it suffices that the measurement operators are orthogonal projection operators, i.e. it holds that $M_i M_j = \delta_{i,j} M_i$ in addition to the completeness condition. This kind of measurement is called a *projective measurement.* It turns out that if we can perform an additional unitary operation before the measurement, this restricted model is equivalent to the general one.

Example 3.4 Perhaps the simplest projective measurement is the *measurement in the computational basis.* There are two interesting observables of the one qbit system of this type:

$$\begin{aligned} O_{\text{proj}} &= \{|0\rangle\langle 0|, |1\rangle\langle 1|\}, \\ O_{\text{Id}} &= \{|0\rangle\langle 0| + |1\rangle\langle 1|\} = \{I\}. \end{aligned}$$

Let us apply the O_{proj} measurement to a qbit. If the qbit is in the superposition state $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$, a simple calculation yields that the probability of observing 0 is $|\alpha|^2$ and the target quantum state after the measurement is $|0\rangle$. Further, the probability of observing 1 is $|\beta|^2$ and the target quantum state after the measurement is $|1\rangle$. On the contrary, the measurement O_{Id} leaves the quantum state untouched and it reveals no information at all.

Composing projective measurement observables of composite systems is straightforward. They can be composed by a tensor multiplication of the individual measurement operators, e.g. if we want to observe only the first qbit of two qbits, we use the observable O_{bit1}, if we want to observe both qbits, we use the observable O_{both}:

$$\begin{aligned} O_{\text{bit1}} &= \{|00\rangle\langle 00| + |10\rangle\langle 10|, |01\rangle\langle 01| + |11\rangle\langle 11|\}, \\ O_{\text{both}} &= \{|00\rangle\langle 00|, |01\rangle\langle 01|, |10\rangle\langle 10|, |11\rangle\langle 11|\}. \end{aligned}$$

O_{both} collapses the two qbit system into a particular basis state. The behaviour of O_{bit1} is more interesting. It collapses the system into a superposition of states consistent with the measurement outcome, leaving their amplitudes untouched except for rescaling. For example, if the system was in the superposition state $\sum_{i=0}^{3} \alpha_i |i\rangle$ immediately before the measurement and the outcome 1 was measured, then the system will collapse to the state $\frac{1}{|\alpha_1|^2 + |\alpha_3|^2}(\alpha_1|01\rangle + \alpha_3|11\rangle)$.

Example 3.5 An important example of an observable in other than computational basis is

$$\begin{aligned} O_{\text{Had}} &= \{\tfrac{1}{2}(|0\rangle + |1\rangle)(\langle 0| + \langle 1|), \tfrac{1}{2}(|0\rangle - |1\rangle)(\langle 0| - \langle 1|)\} = \\ &= \left\{ \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \right\}. \end{aligned}$$

It is nothing else than a projective measurement in the basis $(|0\rangle + |1\rangle)/\sqrt{2}$, $(|0\rangle - |1\rangle)/\sqrt{2}$. We denote it by O_{Had}, because the conversion operator between the computational basis and this one is the Hadamard operator.

We see that measurements reveal very poor information about the original quantum state — the quantum states form a continuum and the measurement outcome is a discrete value. Unfortunately the state is disturbed by the measurement, hence the measurement can not be repeated to reveal more information. The (difficult) task of the quantum complexity theory is developing quantum algorithms that yield the desired information merely from the measurement outcomes.

Note 3.8 One of the essential problems in quantum computing is the problem of distinguishing quantum states. Having a promise that the system is situated in one of the fixed quantum states, our task is determining the quantum state. It turns out that doing this with probability 1 is impossible unless the fixed states are orthogonal. It follows from the completeness condition of the observables. This is also the reason why it is impossible to encode reliably more than 1 classical bit into a qbit — the dimension of the qbit vector space is 2, hence there are only two orthogonal vectors there.

3.2 Quantum Circuits

Perhaps the most natural model of quantum computation is a Quantum Circuit model. It is very well described in [NC00]. We shall mention it very briefly only in this section and concentrate ourselves on building an alternative model of a Quantum Branching Program.

The state of a quantum computer is described by n qbits initialised to $|0\rangle^{\otimes n}$. The evolution is described by a program comprising of a sequence of quantum gates applied to tuples of qbits. There are miscellaneous quantum gates available in various circuit classes QNC $\subseteq$ QAC $\subseteq$ QACC:[4]

- one qbit gates (all of them are available in QNC):
  - the Pauli X, Y, Z gates,
  - the Hadamard gate,
  - a general one qbit gate,

[4] for an incentive discussion of these circuit classes, read [GHMP01, Moo99]

- two qbit gates (all of them are available in QNC):
  - the controlled NOT gate,
  - the swap gate,
  - the phase shift gate,
- three qbit gates (all of them are available in QNC):
  - the Toffoli gate — the NOT operation controlled by a logical conjunction of two qbits,
- n-qbit gates:
  - the n-qbit Toffoli gate — it performs the AND operation of $n-1$ input qbits in unit time (available in QAC),
  - the fanout gate — one qbit controls a bunch of NOT operations on $n-1$ other qbits (available in QACC),
  - the parity gate — a parity of a bunch of control qbits controls the NOT operation on a target qbit (it is equivalent to the fanout gate, thus it is also available in QACC),
  - the Mod[c] gate — a target qbit is flipped if the number of control qbits set to 1 modulo c is nonzero (for $c = 2$ it is a parity gate, it is also available in QACC).

The quantum system is measured in the computational basis at the end of the computation.

We state an interesting relation between the fanout gate and the parity gate as an example in Figure 3.2. The left gate is a fanout gate with the topmost control qbit, the right gate is a parity gate with the topmost target qbit. The right gate is preceded and followed by one qbit Hadamard gates.

3.3 Quantum Turing Machines

Assuming the reader is conversant with classical Turing Machines (they are defined in almost every textbook of computational complexity, for example [Gru97, Pap94, Sip97]), we shall discuss directly their quantum variant. A Quantum Turing Machine model was proposed by J. Watrous in [Wat98]. Let us review his definition and all related concepts.

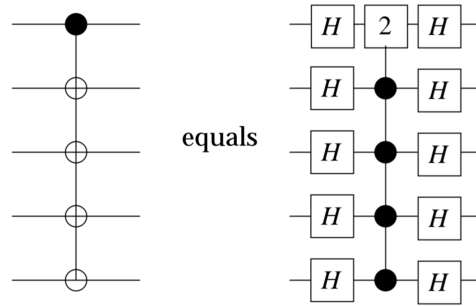


Figure 3.2: Relation between the fanout and parity gates

3.3.1 Definition

Definition 3.2 A *Quantum Turing Machine* (shortcut QTM) is an ordered sequence $M = (\Sigma, \Pi, Q, q_0, \mu)$ such that

- Σ is the input alphabet containing the blank $\# \in \Sigma$,
- Π is the output alphabet not containing the empty word $\varepsilon \notin \Pi$,
- Q is a set of internal states,
- $q_0 \in Q$ is a starting state,
- $\mu : Q \times \Sigma \times \Sigma \times Q \times \{-1, 0, 1\} \times \Sigma \times \{-1, 0, 1\} \times (\Pi \cup \{\varepsilon\}) \to \mathbf{C}$ is a *transition function.* The following restrictions are placed on allowable transition functions:
  - there exist mappings $D_i, D_w : Q \to \{-1, 0, 1\}$ and $Z : Q \to \Pi \cup \{\varepsilon\}$
  - and unitary mappings $V_\sigma : \ell_2(Q \times \Sigma) \to \ell_2(Q \times \Sigma)$ for all $\sigma \in \Sigma$ such that

$$\mu(q, \sigma_i, \sigma_w, q', d_i, \sigma'_w, d_w, \pi) = \begin{cases} \langle q', \sigma'_w | V_{\sigma_i} | q, \sigma_w \rangle & d_i = D_i(q'), \\ & d_w = D_w(q'), \\ & \pi = Z(q'), \\ 0 & \text{otherwise.} \end{cases}$$

The definition is very technical and needs a further explanation. A QTM suited to the study of space-bounded classes has three tapes: a read-only input tape with a two-way tape head, a read-write work tape with a two-way tape head, and a write-only output tape with an one-way tape

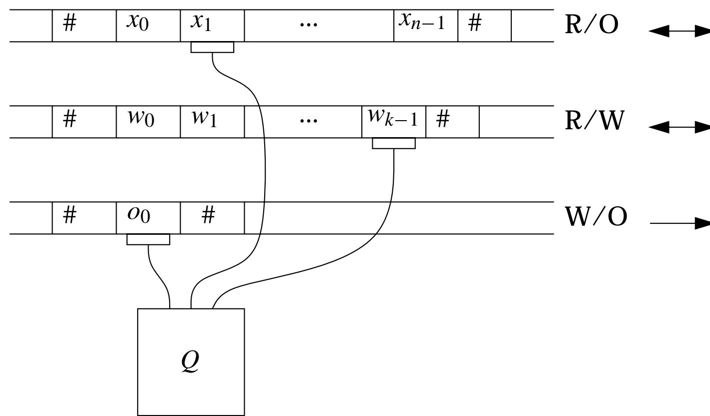


Figure 3.3: Design of a Quantum Turing Machine

head. We assume the tapes are two-way infinite and indexed by $\mathbf{Z}$. See Figure 3.3 for an illustration. The configuration of a QTM comprises:

i. the internal state of the machine,

ii. the position of the input tape head,

iii. the contents of the work tape and the position of the work tape head,

iv. and the contents output tape and the position of the output tape head.

The set of the ***configurations of a QTM*** M is denoted by $C(M)$ or just C if the identity of M is clear from the context. The ***initial configuration*** denoted by $c_0 \in C$ is that one in which the internal state is q_0, all tape heads are positioned over the squares indexed by zero, and all tape squares on the work and output tapes contain blanks #. We assume the input x is written in squares $0, 1, \ldots, |x| - 1$ on the input tape and all remaining squares on the input tape contain blanks #.

As usual in the quantum world, at every instant the state of a QTM can be described by a superposition of configurations $|\psi\rangle \in \ell_2(C(M))$.

Each number $\mu(q, \sigma_i, \sigma_w, q', d_i, \sigma'_w, d_w, \pi)$ may be interpreted as the amplitude with which M, currently in state q, reading symbols σ_i, σ_w on its input and work tapes, will change its internal state to q', move its input tape head in direction d_i, write σ'_w to its work tape, move its work tape head in direction d_w and, if $\pi \neq \varepsilon$, write π on its output tape and move its output tape head to the right (the output tape is untouched when $\pi = \varepsilon$).

3.3.2 Transition function

The conditions placed on the transition function μ look very involved at a first glance. The existence of the mappings D_i, D_w, Z ensures that the moves of all machine heads are fully determined by the target state q', because all contradicting transitions have zero amplitudes. This requirement arises for the sake of the reversibility of the TM, it is not a quantum property yet. Indeed it is a variant of the parental condition from Definition 2.12 of a reversible BP on page 15. If the converse is true, the machine would not know which squares shall it use for the decision of how to perform a reverse step. Further, it would also be possible to construct a non-reversible counterexample similar to the one in Figure 2.2 on page 16.

Let the machine be situated in a classical state (the behaviour in the superposition states is fully determined by the behaviour in the classical states thanks to the linearity). The machine knows both in the forward and reverse direction of computation which input square shall it decide on. Further, the square is placed on a read-only tape hence it is not in a superposition. After the input letter σ_i is read, a proper unitary mapping V_{σ_i} is chosen. According to the definition, the partial configuration vector $|q, \sigma_w\rangle$ is transformed to $|q', \sigma'_w\rangle = V_{\sigma_i}|q, \sigma_w\rangle$ and the tape head movements are determined by the target state q'. Hence the QTM is well-formed — see the next subsection.

Note 3.9 The power of QTM's depends greatly upon the values the transition amplitudes may take. It can be shown that if the general transcendent numbers are allowed, then the non-recursive sets can be recognised. Hence we shall limit them to be algebraic numbers. The choice of rational numbers also suffices for the bounded error case, but it complicates the analysis, since the numbers like $\sqrt{2}/2$ are very commonly used.

3.3.3 Quantum behaviour and language acceptance

Time evolution operator U_x^M of a QTM M for a given input $x \in \Sigma^*$ can be written as

$$U_x = \sum_{c,c' \in C(M)} \alpha(c \vdash c')|c'\rangle\langle c|,$$

where $\alpha(c \vdash c')$ is the amplitude associated with the transition from c to c'. The time evolution operator needs to be unitary. It can be shown that every QTM M obeying the restrictions on transition functions discussed above is well formed, i.e. U_x^M is unitary for every x. We shall not prove it here, but a similar claim for Quantum Networks is proved in Chapter 4.

A QTM is observed after every computational step. The ***observable*** corresponds to simply observing the output tape. For every $w \in \Pi^*$, let P_w be the projection from $\ell_2(C)$ onto the space spanned by classical states for which the output tape contents and the output tape head position are described by w. Now $\{P_w | w \in \Pi^*\}$ is a formal observable of a QTM. Since the QTM is observed after every computational step, the nonempty output word is indeed one letter long.

The ***computation of a QTM*** M on input x proceeds as follows. It starts in the configuration c_0. At every computational step, M is evolved according to U_x and observed as described above. The computation continues until a nonempty word $\pi \in \Pi$ at the output tape has been observed. The result of the computation is the measured π.

For a given QTM M, an input $x \in \Sigma^*$, and $k \in \mathbf{N}$, let $p^M_{\pi,k}(x)$ denote the probability that each observation at the time $k' < k$ yields ε and the observation at time k yields π. It is straightforward to prove by induction that

$$p_{\pi,k}(x) = \| P_\pi (U_x P_\varepsilon)^k |c_o\rangle \|^2 .$$

Let $p_\pi(x) = \sum_{k \in \mathbf{N}_0} p_{\pi,k}(x)$ be the probability that the computation on input x results π. The ***language acceptance*** of QTM's in distinct modes is defined in the same way as the language acceptance of PBP's in Definition 2.9 on page 11.

3.3.4 Complexity measures and complexity classes

Let us define the complexity measures of a QTM. The following information needs to be encoded in the configuration: the internal state, the position of the input and work heads, the contents of the work tape and the first symbol (if any) written to the output tape. The position is encoded in binary form, thus the length of the encoding is logarithmic in the input size and linear in the distance of the furthest non-blank square on the work tape. We say that the space required for a superposition is the maximum space required for any configuration with a nonzero amplitude. We say that a QTM M on input x ***runs in space*** s, if each superposition obtained during the execution of M on input x requires space at most s, i.e.

$$(\forall k \geq 0)(\forall c \in C) \left(\langle c | (U_x P_\varepsilon)^k | c_0 \rangle \neq 0 \Longrightarrow c \text{ requires space at most } s \right).$$

We say that M on input x ***runs in maximal time*** t if

$$\sum_{k \leq t} \sum_{\pi \in \Pi} p_{\pi,k}(x) = 1,$$

i.e. M on input x halts with certainty after no more than t steps of computation. If there exists a finite $t(x)$ for every input x such that M on input x runs in maximal time $t(x)$, we say that M ***halts absolutely***. We can also define the ***expected time*** of the computation of M on input x in the same way as it was defined for a PBP:

$$\text{exp.time}^M(x) = \sum_{k \in \mathbf{N}_0} \sum_{\pi \in \Pi} k \cdot p_{\pi,k}(x).$$

Let $s, t : \mathbf{N} \to \mathbf{N}$ be a space constructible (time constructible respect.) function. We say that a QTM M has the ***space complexity*** s (***time complexity*** t respect.), if for every input $x \in \Sigma^*$, M on input x runs in space $s(|x|)$ (time $t(|x|)$ respect.).

We also define the ***quantum complexity classes*** similarly to the classical ones in Definition 2.11 on page 13.

Definition 3.3 The class $m - Q\text{--}\mathrm{TIME}(t(n))$ ($m - Q\text{--}\mathrm{SPACE}(s(n))$ respect.) is defined as the class of languages recognised by QTM's with time complexity $f(n)$ (space complexity $f(n)$ respect.) in mode m.

$$\begin{aligned}
m - Q\text{--}\mathrm{TIME}(t(n)) = \ & \{L \subseteq \Sigma^* \mid (\exists \text{ QTM } M_L) \\
& M_L \text{ has time complexity } \mathrm{O}(t(n)) \ \& \\
& M_L \text{ accepts } L \text{ in mode } m\}, \\
m - Q\text{--}\mathrm{SPACE}(s(n)) = \ & \{L \subseteq \Sigma^* \mid (\exists \text{ QTM } M_L) \\
& M_L \text{ has space complexity } \mathrm{O}(s(n)) \ \& \\
& M_L \text{ accepts } L \text{ in mode } m\}.
\end{aligned}$$

Note 3.10 There is an interesting gap between the classical and the quantum deterministic classes $Eq\text{--}\mathrm{TIME}(f(n))$ and $Eq - Q\text{--}\mathrm{TIME}(f(n))$. Although quantum computers use something like random numbers during the measurements, not all quantum algorithms are erroneous. It can be shown that there are problems, for which there exists an *exact* quantum algorithm with no classical counterpart, e.g. the Deutsch-Jozsa black-box function algorithm. It is solvable in constant number (one) of queries on a quantum computer but every deterministic algorithm solving it exactly needs to ask an exponential number of queries and every randomised algorithm needs a linear number of queries to achieve an exponentially small error. The classical lower bound is easily proved by the Yao's principle (see [Yao83]). The quantum algorithm is described in [NC00].

However we shall not discuss the famous quantum algorithms here, like the Groover's and Shor's ones. They are also very well described in [NC00].

3.3.5 Nonuniform version

The Turing Machine model is a uniform model in principle. Since we want to prove the equivalence between QTM's and QBP's in both cases, a concept of non-uniformness must be added.

We say that a QTM is a *Quantum Turing Machine with advice*, if the machine can ask for bits of the advice during the computation. The advice does not depend directly on the input $x \in \Sigma^*$, but it depends on its length $|x|$.

Note 3.11 The power of the augmented model seriously depends on the length of the advice. For example, if the advice is permitted to be exponentially long, it can contain the information about every word $x \in \Sigma^n$, whether or not it is present in the language. If the length is smaller, the power is substantially decreased — a good choice of the length limit of the advice could be a polynomial in n, a power of the logarithm of n or a constant. However even the classical TM's with constant length of advice recognise the non-recursive languages in unary encoding (there is only one correct input word for every input size in the unary encoding, hence if the advice contains the solution of the problem, then the TM has just to read it and return it as the result).

We shall define new quantum complexity classes distinguished by the length of the advice. For example $m - Q-\mathrm{TIME}\,(f(n))\,/g(n)$ is the class of languages accepted by QTM's with time complexity $f(n)$ and with advice of length not longer than $g(n)$ in mode m.

Let us extend the definition of a QTM by adding the advice. We introduce a new read-write tape for writing the inquiries. The inquiry tape behaves exactly like another working tape. The transition function μ is properly extended and the restrictions are modified: a new mapping D_q is added, because the tape head movement on this tape must be fully determined by the target state q', and the individual unitary mappings are extended to $V_\sigma : \ell_2(Q \times \Sigma \times \Sigma) \to \ell_2(Q \times \Sigma \times \Sigma)$ for every $\sigma \in \Sigma$.

Having two working tapes instead one does not extend the machine power yet. We still have to provide a special instruction performing the inquiry. We reserve three internal states $q_{\mathrm{hi}}, q_{\mathrm{h0}}, q_{\mathrm{h1}}$ and define the following behaviour: If the machine enters the internal state q_{hi} in a computational basis configuration, it decides on the contents of the inquiry tape and changes its internal state to either q_{h0} or q_{h1} depending on the result of the inquiry. The contents of the tapes and the positions of the tape heads are unchanged. The inquiries have the following format: a binary number denoting the index is read from the inquiry tape and the corresponding bit

of the advice is returned. Further, the computation continues by executing normal program instructions.

The behaviour in superposition configurations is determined by the behaviour in the computational basis configurations thanks to the linearity.

Note 3.12 In simulations discussed in following chapters, the advice will typically contain the encoded description of the simulated nonuniform QBP.

3.3.6 Special forms of QTM's

Definition 3.4 We say that a QTM is *oblivious*, if the movements of all its tape heads are fully determined by the problem size, i.e. they depend neither on the particular input written on the input tape nor on the particular computation path traced.

Note 3.13 The length of the computation of an oblivious QTM is also fully determined by the problem size, since the computation finishes when the output tape head moves first to the right, which happens at the same time in all instances.

Note 3.14 It is well known that every TM can be simulated by an oblivious TM with constant space overhead and quadratic time overhead. We do not show it here. The same method can be used also in the quantum case.

Definition 3.5 We say that a QTM is indeed a *Reversible Turing Machine* (shortcut RTM) iff all amplitudes of the transition functions V_σ are either 0 or 1.

Note 3.15 Since the transition operator must be unitary and all amplitudes are either 0 or 1, it follows that it is a permutation matrix. Hence the RTM is indeed a TM with an additional constraint being reversible.

Chapter 4

Quantum Networks

Having looked at classical Branching Programs and Quantum Turing machines, we shall now propose a model of a Quantum Branching Program. However for the reasons that turn out in Chapter 7 about achieving reversibility, we first propose a generalised model called a Quantum Network and define the Quantum Branching Program as its acyclic variant.

During writing of this document, the model has already been published, see [AGK01, NHK00, SS01].

4.1 Raw Quantum Networks

4.1.1 Definition

Definition 4.1 A *Quantum Network* (shortcut QN) is an ordered sequence $P = (n, \Sigma, \Pi, Q, E, q_0, d, \mu)$ such that

- $n \in \mathbf{N}$ is the length of the input,
- Σ is the input alphabet,
- Π is the output alphabet,
- Q is a set of graph vertices,
- $E \subseteq Q^2$ is a set of oriented edges,
- $q_0 \in Q$ is a starting vertex,
- $d : Q \to \mathbf{Z}_n \cup \Pi$ is a function assigning input variables to internal graph vertices and output results to graph sinks,

- $\mu : E \times \Sigma \to \mathbf{C}$ is a transition function assigning a complex amplitude to the transition through an edge given a letter from the input alphabet.

Let us establish the following notation:

- Q_{inp} denotes the set of input vertices (the sources of (Q,E)),
- Q_{out} denotes the set of output vertices (the sinks of (Q,E)),
- $Q_\pi = \{q \in Q_{\text{out}} | d(q) = \pi\}$ for $\pi \in \Pi$ denotes the set of output vertices assigned to a given result,
- $\overline{Q_{\text{x}}} = Q - Q_{\text{x}}$ is the complement of a given vertex set,
- $Q_{\text{par}} = \overline{Q_{\text{out}}}$ and $Q_{\text{child}} = \overline{Q_{\text{inp}}}$.

Let us extend the transition function μ to transitions between any two vertices given any input letter: we define the operator $\mu_\sigma : \ell_2(Q_{\text{par}}) \to \ell_2(Q_{\text{child}})$ such that

$$\begin{aligned} \langle q'|\mu_\sigma|q\rangle &= \begin{cases} \mu((q,q'),\sigma); & (q,q') \in E, \\ 0; & \text{otherwise}. \end{cases} \\ \Big(\Longleftrightarrow \mu_\sigma &= \sum_{(q,q')\in E} \mu((q,q'),\sigma)|q'\rangle\langle q|\Big) \end{aligned}$$

Then the following requirements must hold:

i. (Q,E) is an oriented graph,

ii. $d(q) \in \Pi$ iff q is a sink of the graph, i.e. $d(Q_{\text{out}}) \subseteq \Pi$ and $d(Q_{\text{par}}) \subseteq \mathbf{Z}_n$,

iii. the parental condition must be fulfilled — for every vertex q there is exactly one input variable assigned to all parents of q:

$$(\exists d_p,\, d_p : Q \to \mathbf{Z}_n)\ (\forall q_p, q \in Q) \quad (q_p,q) \in E \Rightarrow d(q_p) = d_p(q),$$

iv. the operator μ_σ must be unitary (i.e. norm preserving and invertible) for every $\sigma \in \Sigma$.

The first two requirements tell us what a QN looks like. The last two requirements are needed to guarantee that the QN is well-formed, the significance of both of them is demonstrated later. (*The parental condition has already been discussed in Note 2.9 about reversible BP's on page 15.*

The necessity of the parental condition is also supported by Note 4.4 on page 43. The unitarity condition is necessary for physical feasibility of the quantum model, a similar condition has already been seen in Definition 3.2 of a QTM on page 27.)

Note 4.1 When not explicitly given, we assume the input alphabet is taken as $\Sigma = \{0,1\}$, the output alphabet is taken as $\Pi = \{\mathrm{acc}, \mathrm{rej}\}$ and the input size n is chosen automatically according to the labels of internal vertices.

Note 4.2 We assume that the values of the transition function μ are algebraic numbers. As stated in Note 3.9 on page 29, the target model would be incredibly powerful otherwise.

The definition of a QN is powerful in the sense that it allows a general graph with many cycles and components. Let us consider what happens if we restrict the graph layout. Requiring the graph connectivity does not change the power of the QN model, since the computation never gets into another component anyway, but the description of a QN can become more complicated in some cases. However it is obvious that if we require the graph acyclicity, infinite computations are not possible. The infinite loops are not interesting in deterministic models, but they are very useful in probabilistic models. The restricted computation model of a QN with acyclic graph will be called a Quantum Branching Program.

Definition 4.2 We say that $P = (n, \Sigma, \Pi, Q, E, q_0, d, \mu)$ is a *Quantum Branching Program* (shortcut QBP), if it is a Quantum Network, and the following requirements hold:

i. (Q,E) is a connected acyclic oriented graph,

ii. $q_0 \in Q_{\mathrm{inp}}$.

Let us describe the computation of a QN, i.e. let us state what is included into its configuration, how to perform a computation step and how to measure the result: Everything that the QN remembers is comprised in its state. In every step the QN performs the transition corresponding to the value of the input variable assigned to current vertex. Before a transition is performed, the QN is observed whether it is situated in an output vertex. If this happens, the computation stops and the corresponding result is returned.

Definition 4.3 We say that a QN P is in *configuration* c, if it is currently in the state $c \in Q$.

As usual in the quantum world, at every instant the state of a QN may be described by a superposition of configurations $|\psi\rangle \in \ell_2(Q)$.

For a QN $P = (n, \Sigma, _, Q, E, _, d, \mu)$, given input $x \in \Sigma^n$ and a pair of configurations $c, c' \in Q$, the amplitude associated with performing the transition $c \vdash c'$, denoted by $\alpha_x(c \vdash c')$, is defined as $\alpha_x(c \vdash c') = \langle c'|\mu_{x_{d(c)}}|c\rangle$.

Definition 4.4 *Time evolution operator* of a QN $P = (n, \Sigma, _, Q, E, _, d, \mu)$, denoted by $U_x^P : \ell_2(Q) \to \ell_2(Q)$, is defined as

$$U_x = \sum_{c,c' \in Q} \alpha_x(c \vdash c')|c'\rangle\langle c|,$$

hence if the program P on input x is situated in superposition $|\varphi\rangle$, then after unobserved evolving for one step its new superposition will be $U_x|\varphi\rangle$.

Definition 4.5 *Observable* of a QN P is defined as

$$\{P_\pi | \pi \in \Pi \cup \{\text{work}\}\},$$

where $P_\pi = \sum_{c \in Q_\pi} |c\rangle\langle c|$ is a projection from Q onto the space spanned by the given vertex set. The sets Q_π for $\pi \in \Pi$ have already been defined and $Q_{\text{work}} = Q_{\text{par}} = \overline{Q_{\text{out}}}$.

Definition 4.6 The *computation* of a QN P on input x proceeds as follows:

1. The QN P is situated to the classical state $|q_0\rangle$.
2. The following loop is executed until an output result is observed. Each step of the computation consists of two phases:
   - 2.1. the state of program is observed as described above,
   - 2.2. the program evolves according to U_x.
3. The observed result is returned.

4.1.2 Consistency of the model

We say that a QN P is *well-formed*, if for every input x its time evolution operator U_x restricted to $\ell_2(Q_{\text{par}}) \to \ell_2(Q_{\text{child}})$ is unitary and if the observable of P is well defined.

Theorem 4.1 Every QN (satisfying the requirements in Definition 4.1) is well-formed.

The outline of the proof: We shall show, that the matrix of every restricted time evolution operator U_x is a two dimensional permutation[1] of a block-diagonal matrix. A block-diagonal matrix is unitary iff each block incident with the main diagonal is unitary. The diagonal blocks are independent, thus every such block can be replaced by any other unitary block without breaking the unitarity of the whole operator. The parental condition implies that the program decides on exactly one input variable in every such block. Thus the evolution operator for any input can be combined from the blocks of the unitary operators μ_σ and hence it is also unitary.

For a QN $P = (_, \Sigma, _, _, E, _, _, \mu)$ and an edge $e \in E$, we say that the edge e ***has colour*** $\sigma \in \Sigma$, if $\mu(e,\sigma) \neq 0$. An edge can have more colours. The ***set of edges having colour*** σ, denoted by E_σ, is defined straightforwardly as $E_\sigma = \{e \in E | \mu(e,\sigma) \neq 0\}$, the corresponding graph (Q, E_σ) will also be called ***monochromatic.***

For an oriented graph (V,E) and an ordered pair (P,C) of sets $P,C \subseteq V$, we say that (P,C) is the ***family of parents P and children C*** if for every parent vertex $p \in P$ all its graph children are also in C and vice versa, i.e. $(\forall (p,c) \in E)\ (p \in P \Leftrightarrow c \in C)$. We say that a family (P,C) is ***indivisible***, if it is minimal in inclusion, i.e. every pair (P',C') of proper subsets $P' \subsetneq P$, $C' \subsetneq C$ is not a family. For an ordered pair (P',C'), the ***family induced by*** (P',C') is the minimal family (P,C) such that $P' \subseteq P$ and $C' \subseteq C$. The set of edges belonging to a family is called an ***edge family.***

Lemma 4.2 Let $P = (_, \Sigma, _, Q, E, _, d, \mu)$ be a QN with the parental and unitarity conditions omitted. Let (P,C) be an indivisible family from graph (Q,E) for $P \subseteq Q_{\mathrm{par}}$ and $C \subseteq Q_{\mathrm{child}}$. Then the following holds:

i. any two vertices $v_1, v_2 \in P \cup C$ are connected by a path where the parents and the children are alternated,

ii. if, in addition, the parental condition holds, then all parents $p \in P$ are assigned to the same input variable,

iii. if, in addition, the unitarity condition holds, then $\#P = \#C$.

Proof.

i. Let v_1 be a parent. The family (P,C) induced by $(\{v_1\}, \emptyset)$ can be obtained by adding all children of v_1 into C, followed by adding all parents of every children $c \in C$ into P, ... until the sets P,C are closed.

[1] i.e. both the rows and the columns are independently permuted

v_2 must lie in $P \cup C$ else (P,C) is not indivisible. Hence there exists a path $v_1 - c_1 - p_2 - c_2 - \ldots - v_2$. The proof for the case when v_1 is a child is analogous.

ii. The parental condition implies that all parents of a child are assigned to the same input variable. Every pair of parents is connected by an alternating path and every two adjacent parents in the path are covered by the parental condition, hence all parents are assigned to the same input variable.

iii. The family (P,C) is closed (there is no edge going out of the family of any colour), $P \subseteq Q_{\text{par}}$ and $C \subseteq Q_{\text{child}}$, hence the sub-matrix of every unitary operator μ_σ from Definition 4.1 of a QN induced by $C \times P$ is surrounded by zeroes[2] and consequently μ_σ restricted to $\ell_2(P) \to \ell_2(C)$ must stay unitary. Unitary operators necessarily operate on sets of equal size (else they could not be reversible). Thus $\#P = \#C$.

□

Lemma 4.2 also holds for the *monochromatic* graph (Q, E_σ) for every $\sigma \in \Sigma$. In the third step of the proof, it does not matter which unitary operator μ_σ is used.

Lemma 4.3 Let $P = (_, \Sigma, _, Q, E, _, d, \mu)$ be a QN and $p \in Q_{\text{par}}$ ($c \in Q_{\text{child}}$ respect.). Then for every colour $\sigma \in \Sigma$ there is an edge going from p (going to c respect.) of colour σ.

Proof. Let us take a family (P,C) from graph (Q, E_σ) induced by $(\{p\}, \emptyset)$. Then $\#C = \#P \geq 1$, thus there is an edge of colour σ going from p. The proof for c is analogous. □

This implies that there are no partial output vertices in a QN. Every vertex is either a sink and has no outgoing edges or there is an outgoing edge for every colour. The same holds for ingoing edges.

Lemma 4.4 For any graph (Q,E) the set of edges E can be decomposed into a disjoint union of indivisible edge families.

Proof. A family can also be induced by an edge, we define that the edge (p,c) induces the same family as the vertex sets $(\{p\},\{c\})$ do. Let us take a relation on the edges in which e_1 is related with e_2 iff the families induced

[2] i.e. all other items in the rows spanned by C and all other items in the columns spanned by P are zero

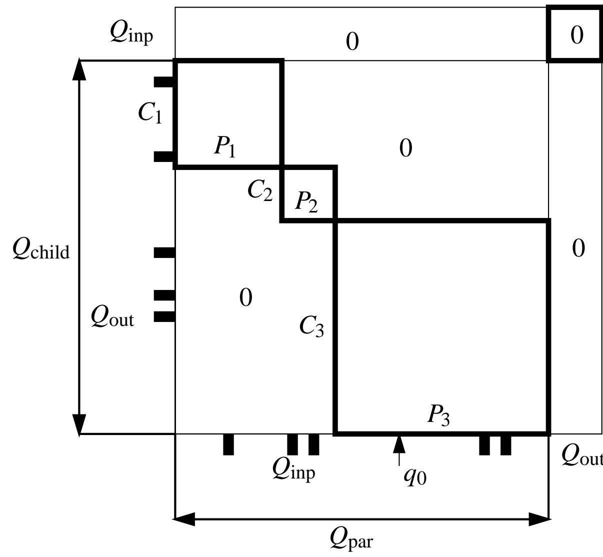


Figure 4.1: Structure of a raw QN

by e_1 and e_2 are equal. The relation is an equivalence, hence there exists a desired decomposition. □

Having the edge decomposition, we can take the parents and the children of the equivalence classes separately. Consequently, for any QN, the states Q_{par} and Q_{child} can be independently reordered according to the equivalence classes in such a way, that for every input x, the time evolution operator U_x is a block-diagonal matrix and the parents of every block are assigned to a unique input variable. All this is illustrated in Figure 4.1.

μ_0	2,1	2,2	3,1	3,2	3,3	1,1
1,1	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$				
1,2	$\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{2}}{2}$				
2,1			1	0	0	
2,2			0	1	0	
2,3			0	0	1	
3,1						1

μ_1	2,1	2,2	3,1	3,2	3,3	1,1
1,1	0	1				
1,2	1	0				
2,1			0	1	0	
2,2			0	0	1	
2,3			1	0	0	
3,1						1

Table 4.1: Transition functions μ_0, μ_1 of the QN in Figure 4.2

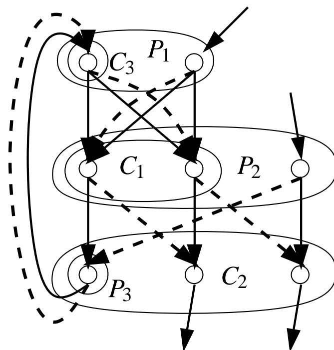


Figure 4.2: Example of a QN

Note 4.3 This decomposition tells nothing about the set of input vertices, because the Lemma 4.2 holds only for indivisible families. The input vertices do not form a special family in general, but they can be mixed with internal vertices in many ways as seen in Figure 4.2. However we conclude that:

Lemma 4.5 The number of input vertices equals the number of output vertices in every QN.

Proof. The unitarity of μ_σ implies that $\#Q_{\text{par}} = \#Q_{\text{child}}$. We know that $Q_{\text{par}} = \overline{Q_{\text{out}}} = Q - Q_{\text{inp}}$ and $Q_{\text{child}} = \overline{Q_{\text{inp}}} = Q - Q_{\text{inp}}$, hence $\#Q_{\text{out}} = \#Q_{\text{inp}}$. □

Proof of Theorem 4.1 The proof of unitarity is simple now: Each restricted time evolution operator U_x is a two dimensional permutation of a block-diagonal matrix. The block-diagonal matrix is unitary iff every its block is unitary. We know that every transition operator μ_σ is unitary and it has the same block structure (except for that the families may not be indivisible, which does not matter). We can replace any diagonal block by another unitary block, and since the parents in every block are assigned to a unique input variable, we may compose the blocks of μ_σ into U_x — every block comprising of parents assigned to an input variable x_i will be replaced by the corresponding block from μ_{x_i}. The resulting matrix becomes exactly U_x.

Yet one more claim must be verified: the correctness of the observable of the QN. We know that $\{Q_{\text{x}} | x \in \Pi \cup \{\text{work}\}\}$ is a decomposition of Q. Hence it is apparent that the projection P_x defined as $P_x = \sum_{c \in Q_x} |c\rangle\langle c|$ fulfils the conditions $P_x^\dagger P_x = P_x^2 = P_x$, $\sum_x P_x = I$ and $P_x P_y = \delta_{x,y} P_x$ and thus the observable is a projective measurement. □

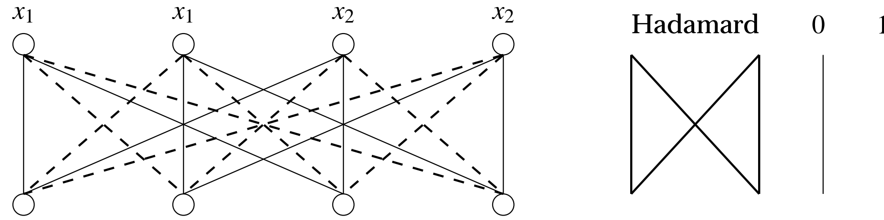


Figure 4.3: A well-formed QBP not fulfilling the parental condition

Note 4.4 The parental condition is needed for the proof. If it is omitted, it would be possible to compose the operators μ_σ into U_x without preserving the unitarity. Recall the counterexample in Figure 2.2 on page 16. On the other hand we must admit that though the parental condition is sufficient it is indeed not necessary. In Figure 4.3, there is a simple counterexample of a layered QBP that does not fulfil the parental condition but it is unitary for all choices of input variables x_1, x_2. It can be proved by simply considering all possibilities, since there are only four of them.

Note 4.5 The transition functions and also the restricted time evolution operator have an interesting property: they all can be naturally extended from $\ell_2(Q_{\mathrm{par}}) \to \ell_2(Q_{\mathrm{child}})$ to $\ell_2(Q) \to \ell_2(Q)$ preserving the unitarity. As you can see in Figure 4.1, the right upper block can be replaced by any unitary matrix, e.g. by a permutation matrix. This would violate the condition of graph acyclicity (for a QBP) and cause the disappearance of Q_{inp} and Q_{out}, but it is noticeable anyway.

When constructing a QN we have to verify all requirements from Definition 4.1. The only nontrivial requirement is the unitarity of every transition operator μ_σ. It is easy to prove a local condition of unitarity now:

Theorem 4.6 The transition operator $\mu_\sigma : \ell_2(Q_{\mathrm{par}}) \to \ell_2(Q_{\mathrm{child}})$ is unitary iff the following conditions hold for every family (P,C) induced by $(\{p\},\emptyset)$ or $(\emptyset,\{c\})$, for $p \in Q_{\mathrm{par}}$, $c \in Q_{\mathrm{child}}$:

i. $\#P = \#C \geq 1$,

ii. μ_σ restricted to $\ell_2(P) \to \ell_2(C)$ is unitary.

Proof. Straightforward. □

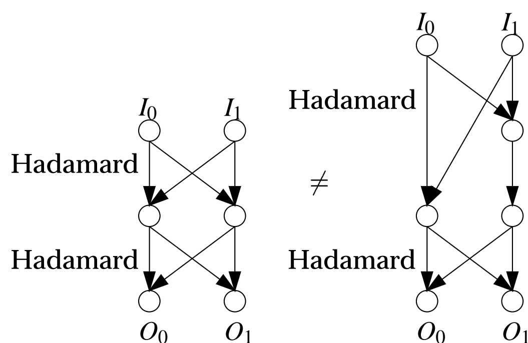


Figure 4.4: The interference pattern of a QBP

4.1.3 Uniform sequences of QN's

First of all, we shall define the sequence of QN's in the same way as in Definition 2.4 of the sequence of BP's on page 7.

Definition 4.7 A *sequence of Quantum Networks* P is an infinite sequence $P = \{P_i\}_{i=1}^{\infty}$ of QN's where for every input size a special network is provided.

A sequence of QN's alone is a nonuniform model in principle. The concept of uniformness is greatly inspired by Definition 2.7 of the uniform sequence of BP's on page 8, however some notions must be substantially changed to obtain a model equivalent to the QTM model, which is the representative of uniform computational models. Let us look at a QTM. Since a QTM has a finite number of internal states, it has also a finite number of transition types. We shall require the same from the sequence of QN's being simulated.

While in the classical probabilistic case it was possible to circumvent this impediment (and not require this requirement) by simply simulating the complex random choice types by the chain of the simple ones (see Note 2.7 on page 14), the same method completely fails in the quantum case for a reason inherent to quantum computation — the quantum interference. If we can not ensure that every computational path takes the same time in the simulation, the new interference pattern seriously changes the program behaviour (see Figure 4.4). In fact we can not ensure that, because when there are infinitely many distinct transition types, they can not be simulated by a fixed set of transitions in fixed finite number of steps by any algorithm (proved by a simple counting argument).

Example 4.1 Let us scrutinise Figure 4.4. For simplicity we have chosen the input alphabet $\Sigma = \{0\}$. Both QBP's consist of two Hadamard operations, but the computation is delayed for one computational step at one state in the right QBP. The Hadamard operation is self-inverse.

The initial state of both programs is $|I_0\rangle$. The left QBP stops after two time-steps and reaches O_0 with probability 1. The right QBP stops after two or three time-steps with probability $1/2$ and it always reaches both sinks with probability $1/2$.

Definition 4.8 For a QN $N = (_, \Sigma, _, Q, E, _, _, \mu)$, an indivisible family (P, C) and an input letter $\sigma \in \Sigma$ the ***transition type*** of family (P, C) and letter σ is the (unitary) matrix $\mathbf{type}((P,C),\sigma) : C \times P \to \mathbf{C}$ taken as the representation of the restricted operator μ_σ in the computational basis:[3]

$$(\forall p \in P, c \in C) \;\; \mathbf{type}((P,C),\sigma)_{c,p} = \langle c|\mu_\sigma|p\rangle.$$

Let $\mathbf{closure}_G(P',C')$ denote the family induced by (P',C') in graph G. We say that a sequence of QN's $N = \{N^{(n)}\}_{n=1}^{\infty}$ has ***finite number of transition types***, if the set of transition types in monochromatic graphs for all input sizes is finite, i.e.

$$\left| \bigcup_{\sigma\in\Sigma} \bigcup_{n\in\mathbf{N}} \bigcup_{q\in Q^{(n)}_{\mathrm{par}}} \left\{ \mathbf{type}\left(\mathbf{closure}_{(Q^{(n)},E^{(n)}_\sigma)} (\{q\},\emptyset), \sigma \right) \right\} \right| < \infty.$$

Definition 4.9 Let P be a QN with some fixed numbering of its transition types. The ***encoding of QN*** $P = (n, \Sigma, \Pi, Q, E, q_0, d, \mu)$ is defined in the following way:

- the vertices are uniquely identified by natural numbers,[4] $q^{\max}$ be the maximal allocated vertex number,
- for every vertex $q \in Q$ a pair $(d(q), d_p(q))$ of indexes of input variables the computation decides on is formed (the first item in the pair serves for the forward direction of computation, the second one serves for the backward direction of computation), if $q \in Q_{\mathrm{out}}$, then the identifier of the output result produced is stored instead,

[3]Notice that for every family size $f = \#P$ the same operator can be in general represented by $f!^2$ distinct matrices corresponding to the distinct permutation of parents and children. However it does not matter since $f!^2$ is a finite number.

[4]not necessarily in consecutive way

- for every input letter $\sigma \in \Sigma$ the monochromatic graph (Q, E_σ) is decomposed into distinct indivisible families, identified also by natural numbers, $F_\sigma^{\max}$ be the maximal allocated family number,
- every family carries an index of its transition type (which indeed comprises also the family size),
- for every input letter σ two sorted translation tables are formed, they translate the vertex number to the pair [*family identifier, index of the vertex in the list of family parents/sons*] in corresponding monochromatic graph, the reverse translation tables are also present,
- the description of a QN consists of:
  - a header (comprising of the input size n, input alphabet size $\#\Sigma$, max. vertex identifier $q^{\max}$, identifier of the starting vertex q_0, $\#\Sigma$ numbers of max. family identifiers in the monochromatic graphs $F_\sigma^{\max}$),
  - an array of vertex decision variables (since the vertices need not be indexed in consecutive way, the holes between the records can contain any garbage),
  - $\#\Sigma$ arrays of family transition types,
  - and $4 \cdot \#\Sigma$ translation tables also stored as arrays.

The description of the transition types is not included in the encoding of P, it is regarded as a fixed entity here.

We know that the number of families is less than the number of vertices. Notice that under an assumption that there are not too much holes between the records the length of encoding of a QN in records is asymptotically equal to the number of vertices of the QN up to a multiplication constant. This will be useful for computing the length of the advice of a QTM.

Definition 4.10 A sequence of QN's $P = \{P_n\}_{n=1}^{\infty}$ is called *uniform* if the following requirements hold:

- P has a finite number of transition types, we can index these with natural numbers,
- there exists a RTM M that for given $n, b \in \mathbf{N}$ yields the b-th bit of the encoding of P_n in that fixed numbering of transition types. Both inputs n, b are expected to be written in binary form using $\lceil \log_2 n \rceil$

bits on the input tape. The output of M is returned in its internal state and the work tapes must be cleaned at the end of the computation. The length of computation of M must be independent on b, i.e. for given n the length of computation is constant among all b's.

Such machine M is called a ***construction machine*** of P. If there is no such machine, we call the sequence ***nonuniform***.

Note 4.6 The constraints on the construction machine C are chosen properly to simplify the forthcoming quantum simulations. The reversibility is required since everything must be reversible in quantum world. The constant length of computation is required for preserving the interference pattern. Cleaning the work tapes is required for the ability of repeating calls.

Note 4.7 It is important to understand, why we have chosen, that the decomposition to families operates on individual monochromatic graphs instead of letting it operate directly on the global graph. The latter encoding would also make sense, but the power of the model would be more bounded. A simple example of a RBP in Figure 2.3 on page 16 shows that families of size q arise in the global graph when implementing the operation increment modulo q, while the families in the corresponding monochromatic graphs consist of a simple edge only. Hence a sequence of QN's computing the sum $\sum_{i=1}^{n} x_i$ for input size n (and computing, for example, whether it is greater than say $n/2$) in this simplest way would not be uniform in that model.

Note 4.8 We have based the notion of uniformness on the framework of equivalence with the QTM model. It is also possible to base the uniformness on pure computability, i.e. the ability of constructing effectively the description of the QN including transition probabilities. A model uniform in this sense is not simulated by a QTM in general. In some special cases it can happen that the approximation of a general unitary operation by a chain of fixed operations does not disturb the interference pattern, e.g. when the operations of equal complexity are placed in one layer like in quantum circuits, however if this in not the case then the simulation is not possible.

Example 4.2 A so-called *Quantum Fourier Transform* on n qbits defined as

$$F_n = \left(\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \sum_{j=0}^{2^n-1} \xi^{ij} \langle j| \right), \text{ where } \xi = e^{i2\pi/2^n}$$

can be efficiently implemented by a depth n^2 quantum circuit comprising just of controlled rotations $R_z(2\pi/2^k)$. Such circuit can be represented by a QN of a regular layout. Though this sequence of QN's is not uniform, it can be simulated within arbitrary accuracy by a QTM.

4.1.4 Language acceptance and complexity measures

The computation of a QN consists of many measurements, so it is a probabilistic process. Let $p^P_{\pi,k}(x)$ denote the probability that if a QN P is run on input x as described above, it halts exactly after k steps and the result is π. This condition is equal to yielding "work" at every observation step $k' < k$ and yielding π at observation step k. A straightforward proof by induction shows that

$$p^P_{\pi,k}(x) = \| P_\pi (U_x P_{\text{work}})^k |q_0\rangle \|^2 .$$

Definition 4.11 For a given QN $P = (n, \Sigma, \Pi, Q, E, q_0, d, \mu)$, an input $x \in \Sigma^n$ and a result $\pi \in \Pi$, the ***probability that** P **yields** π **on input** x*, denoted by $p^P_\pi(x)$, is defined as

$$p^P_\pi(x) = \sum_{k=0}^{\infty} p^P_{\pi,k}(x).$$

The superscript P may be omitted when it is clear which P is meant from the context. The same notation is used also for the sequences of QN's.

Definition 4.12 Let $P_1, P_2 = (n, \Sigma, \Pi, _, _, _, _, _)$ be the QN's with the same interface (the input alphabet, the length of input word and the output alphabet). We say that P_1 ***is equivalent to*** P_2 if the probability of yielding $\pi \in \Pi$ is the same for every input $x \in \Sigma^n$, i.e.

$$(\forall x \in \Sigma^n)\ (\forall \pi \in \Pi) \quad p^{P_1}_\pi(x) = p^{P_2}_\pi(x).$$

The same concept is also used for the sequences of QN's.

The ***language acceptance*** of a sequence of QN's in distinct modes is defined in the same way as for the PBP's in Definition 2.9 on page 11.

Definition 4.13 Let P be a sequence of QN's, $L \subseteq \Sigma^*$ be a language, m be a mode listed in Table 2.1 on page 12. We say that ***P accepts L in mode m*** iff for every $x \in \Sigma^*$ the probability $p^P_1(x)$ of yielding result 1 fulfils the appropriate condition listed in the table.

Note 4.9 Using the concept of language acceptance of a sequence of QN's, we could also have defined the that two QN's P_1, P_2 are equivalent if they

induce the same language in a given mode m. The condition stated in Definition 4.12 is stronger in general since every two QN's with equal probabilities of corresponding outputs necessarily induce the same language in every mode.

Let us define the complexity measures for a QN, i.e. the time complexity and the space complexity. They both are defined for a fixed QN at first, then the definition is generalised to sequences of QN's. We use the same approach as the Note 2.2 about the complexity measures of a PBP on page 11.

Definition 4.14 Let P_n be a QN. We define the ***size*** of P_n as the number of its vertices $s = \#Q$ and the ***space complexity*** of P_n as $\lceil \log_2 s \rceil$. Let $s : \mathbf{N} \to \mathbf{N}$ be a space constructible function. We say that a sequence of QN's $P = \{P_n\}_{n=1}^{\infty}$ has the ***space complexity*** s, if for every input size n the QN P_n has the space complexity $s(n)$, and we denote it by $\mathrm{Space}\,(P) = s$.

Definition 4.15 Let P_n be a QN. We say that P_n on input $x \in \Sigma^n$ ***runs in maximal time*** t if

$$\sum_{k \le t} \sum_{\pi \in \Pi} p_{\pi,k}(x) = 1,$$

i.e. P_n on input x halts with certainty after no more than t steps of computation. We can also define the ***expected time*** of the computation of P_n on input x in the same way as for a PBP as

$$\mathrm{exp.time}^{P_n}(x) = \sum_{k \in \mathbf{N}_0} \sum_{\pi \in \Pi} k \cdot p_{\pi,k}(x)$$

and say that P_n ***runs in expected time*** t if $\mathrm{exp.time}(x) \le t$.

Let $t : \mathbf{N} \to \mathbf{N}$ be a time constructible function. We say that a sequence of QN's $P = \{P_n\}_{n=1}^{\infty}$ has the ***maximal time complexity*** t (***expected time complexity*** respect.), if for every input $x \in \Sigma^*$, $P_{|x|}$ on input x runs in maximal time $t(|x|)$ (expected time $t(|x|)$ respect.), and we denote it by $\mathrm{MaxTime}\,(P) \le t$ ($\mathrm{ExpTime}\,(P) \le t$ respect.). If for every input x there exists a finite $T(x)$ such that $P_{|x|}$ runs on input x in maximal time $T(x)$, then we say that P ***halts absolutely***.

Definition 4.16 Let P_1, P_2 be two equivalent sequences of QN's with space complexities s_1, s_2 and time complexities (take either maximal or expected) t_1, t_2.[5] The ***space overhead*** and the ***time overhead*** of P_2 over P_1 are the mappings $o_s, o_t : \mathbf{N} \to \mathbf{N} \cup \{\infty\}$ such that $o_s(n) = s_2(n)/s_1(n)$, $o_t(n) = t_2(n)/t_1(n)$.

[5] meaning the minimal function such that the sequence of QN's still has such time complexity (because the time complexity is defined as an upper bound)

Having defined both the language acceptance criteria and the complexity measures of a sequence of QN's, we could establish a new hierarchy of complexity classes according to this computational model. However we will not do that. We shall rather show that the QTM's and the QN's simulate each other under some restrictions at low cost, hence the classes would be equivalent up to the simulation cost.

If we indeed wished to define them we would do it exactly in the same way as in Definition 2.11 of classical complexity classes on page 13.

4.2 Special forms of QN's

The definition of a QN seems to be too general for some purposes. Let us examine a few models with simpler and more regular inner structure. We shall demonstrate in later chapters that these models are indeed as powerful as the raw model, i.e. every raw QN can be converted into an equivalent regular model with small time and space overhead.

4.2.1 Layered QN's

The first aspect of the graph of a QN we look at is its layout. The computational state of a raw QN can spread unpredictably over large part of the graph in short time. If the vertices are ordered into distinct layers, the computation would be more regular.

Definition 4.17 We say that a QN $P = (_, _, _, Q, E, q_0, _, _)$ is *layered* if there exists a decomposition of vertices Q into layers $Q_0, Q_1, Q_2, \dots, Q_k$ such that

i. $Q_{\mathrm{inp}} \subseteq Q_0$,

ii. $Q_{\mathrm{out}} \subseteq Q_k$,

iii. $(\forall (q,q') \in E) \quad (q \in Q_k \;\&\; q' \in Q_0) \vee (\exists i \in \mathbf{N}_0)\ (q \in Q_i \;\&\; q' \in Q_{i+1})$.

We say that a QBP P is layered, if P taken as a QN is layered and if there are no edges going from the last layer into the first one, i.e. the third condition is replaced by

$$(\forall (q,q') \in E) \quad (\exists i \in \mathbf{N}_0)\ (q \in Q_i \;\&\; q' \in Q_{i+1}).$$

Note 4.10 This layout of a QN is ideal for reducing the number of qbits needed to store and for avoiding the measurement during the computation, since we know at every instant in which layer all states with non-zero

amplitudes are. Thus the information about the current layer needs not be stored as an (expensive) qbit, but it suffices to remember it in a classical external program counter. Reducing the quantum space complexity helps a lot with constructing a physically feasible computer. The measurement has to be be done only in every $(k+1)$-th step.

If the layer number is not represented within the quantum state, the state space of the QN is spanned by the vertices from a fixed layer, not by all graph vertices. It means that the the computational state is not translated into the next layer but it stays within that fixed layer in every computational step. Moreover we have to provide an extra unitary transformation $U_x^{(l)}$ for every layer l. It does not matter, since applying distinct simpler operations in every step is more natural than applying the same complex operation U_x again and again during all the computation, e.g. this approach is nearer to the notion of quantum circuits which also operate in layers. It can also happen that the real number of distinct operators $U_x^{(l)}$ that need to be prepared is indeed decreased, because a layer operator operates on fewer vertices with possibly fewer number of input variables they decide on.

Example 4.3 A fictive layered QN is shown in Figure 4.5. For simplicity the final layer operators $U_x^{(l)}$ are displayed instead of all μ_σ's and the amplitudes are not highlighted. The quantum space complexity is $\log_2 8 = 3$ qbits, the layer counter can be stored classically in $\log_2 4 = 2$ bits, the measurement has to be done in the beginning of every 4-th computational step. The translation from the last layer into the first one is done by a unitary operator of a smaller rank, since the input and output vertices are excluded.

Note 4.11 The concept of the layered layout does not restrict QN's very much. There is a trivial conversion procedure transforming a QN to a layered one, see Section 6.1. However transforming a QBP is a bit more tricky thanks to the acyclicity of the graph.

4.2.2 Oblivious QN's

Next step after converting a QN to a layered layout is reducing the number of variables the computation decides on in every layer.

Definition 4.18 We say that a layered QN $P = (_, _, _, Q, E, _, d, _)$ is *oblivious* if there is exactly one input variable assigned to every layer except for

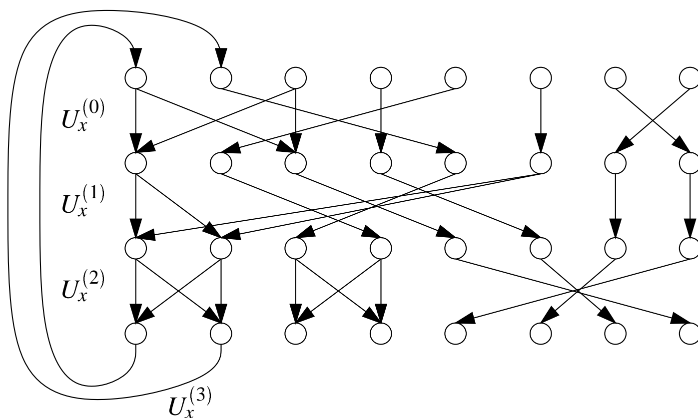


Figure 4.5: Example of a layered QN

the last one. The last layer could have no variable assigned to it if it has no outgoing edges: $((\forall i < k)\ \#d(Q_i) = 1)\ \&\ \#(d(Q_k) - \Pi) \leq 1$.

Note 4.12 This restriction enables reducing the number of distinct unitary transformations that need to be prepared. If the layered QN is oblivious, every operator $U_x^{(l)}$ is indeed equal to one of the $\mu_\sigma^{(l)}$, since the decision variable is equal in all parent vertices in every layer.

4.2.3 Bounded degree QN's

Implementing quantum operations on a large scale becomes physically infeasible. Indeed only small set of discrete operations can be performed. The simplest arrangement we can do is bounding the degree of the graph vertices.

Definition 4.19 We say that a QN $P = (_, \Sigma, _, Q, E, _, _, \mu)$ has ***degree bounded by*** b if for every input letter $\sigma \in \Sigma$ both the input and output degree of every vertex in the monochromatic graph (Q, E_σ) are less or equal to b:

$$(\forall \sigma \in \Sigma)\ (\forall q \in Q) \quad d^-_{(Q,E_\sigma)}(q) \leq b\ \&\ d^+_{(Q,E_\sigma)}(q) \leq b.$$

Note 4.13 This restriction is indeed quite weak, since families of arbitrary size can be arranged from vertices of bounded degree. Moreover even if we bound also the family size, the number of distinct transition types can still be infinite. A much stronger requirement is, for example, the condition of uniformness. However we leave the definition here, since it is a transparent criterion for checking QN's.

Chapter 5

Equivalence of QN's and QTM's

We have defined the computational model of sequences of Quantum Networks. We shall show that, under some circumstances, it is equivalent to the model of Quantum Turing Machines in both nonuniform and uniform case. Moreover both simulations can be done at low cost — polynomial time and polylogarithmic space.

Sequences of Quantum Branching Programs are indeed equivalent to Quantum Turing Machines that never enter a loop.

5.1 Converting a QTM to a sequence of QN's

This section is concentrated on the proof of the following theorem.

Theorem 5.1 Every QTM M with computable space complexity s can be simulated by a uniform sequence of QN's P. The space and time complexities are preserved by the simulation. The construction machine of P runs in space linear in $s(n)$ and in time polynomial in $s(n)$.

Let us define exactly the concept of the computability of functions. This restriction is essential for the uniformness of the target sequence of QN's.

Definition 5.1 We say that a space complexity $s : \mathbf{N} \to \mathbf{N}$ of a QTM is *computable*, if there exists a mapping $s' : \mathbf{N} \to \mathbf{N}$ such that $s(n) \leq s'(n)$ & $s'(n) = \mathrm{O}(s(n))$ and the value $s'(n)$ is computable by a RTM operating in space linear in $s(n)$ and in time polynomial in $s(n)$. Let us also suppose $s(n) = \Omega(\log n)$.[1]

[1] otherwise the space bounds in this section would have to be recomputed

Note 5.1 Remind Definition 4.12 of equivalence of distinct programs on page 48. When simulating a program by another one (possibly in another computational model) we require both programs being equivalent, otherwise the simulation would make no sense.

Proof of Theorem 5.1 The QTM M has bounded space complexity, thus there is a finite number of reachable configurations for every input size n. The QN P_n will be represented by a graph with vertices corresponding to the reachable configurations of M. The edges of the graph of P_n will correspond to the transitions of M between the configurations. The computation of P_n obviously follows the computation of M and the space and time complexities are preserved.

However we have to show, in addition, that the target sequence P is uniform. We known that the source machine M is kind of regular (it has a finite number of internal states and the movement of tape heads is determined by the destination state) and that its space complexity is computable. Let us show that both requirements of uniformness are fulfilled — P has a finite number of transition types and the description of P_n can be constructed by a RTM with constant length of computation.

Transition types: Let us fix an input letter $\sigma_i \in \Sigma$, ignore the tape head movements, and take a monochromatic graph $G^M_{\sigma_i}$ of the transitions of M when the input letter σ_i is read. Vertices of $G^M_{\sigma_i}$ correspond to pairs (q, σ_w), where $q \in Q$, $\sigma_w \in \Sigma$. From the unitarity condition of the QTM M, we see that the graph has the same structure as the monochromatic graph of a QN, i.e. it can be decomposed into distinct families. Moreover the number of families is constant for all input sizes n. Let us index distinct families by natural numbers, e.g. by the number of lexicographically first vertex belonging to it.

Let us imagine the monochromatic graph of the whole QTM M. Without the tape movements, it comprises of infinitely many copies of this graph. Since the tape head movements are fully determined by the target state, these instances are linked together in a regular way like in Figure 5.1. The transitions are permuted, hence it does not happen that, for example, two distinct target vertices are mapped to the same vertex. Hence the global graph of M has the same transition types as $G^M_{\sigma_i}$. The global graph of M is infinite, however we know that the actual space complexity of M is bounded and therefore the subgraph of reachable configurations is also finite.

Vertices of the target network P_n will be indexed by the pair [*subgraph identifier, identifier of the vertex in the subgraph*]. From the structure of the vertex identifier, we see that the identifier according to a configuration

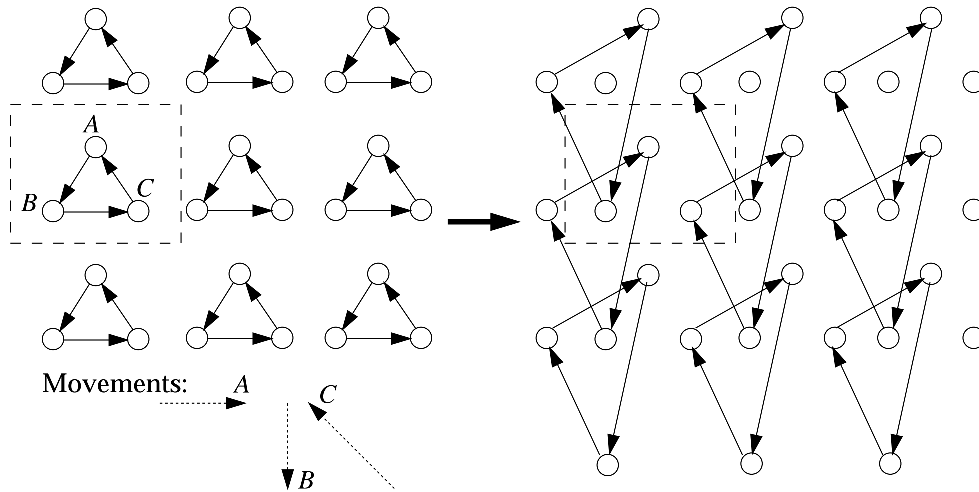


Figure 5.1: Structure of the monochromatic graph of a QTM

of M has the form [*internal state* q, *current working tape letter* σ_w, *the rest of the configuration*]. The identifiers are not necessarily allocated in consecutive order — it can happen that there are some holes at the side subgraphs, again look at Figure 5.1 for an example. The fact that some identifiers do not correspond to a reachable vertex, does not matter, since they can not appear in a consistent simulation. The families are indexed in the same way.

Computability of the encoding of P_n: We are given n, b and we have to compute the b-th bit of the encoding of P_n. Each such computation starts with computing the header information, it then transforms b to the specification of the record asked for, and it finishes by computing and returning the desired bit.

In the beginning, we compute the upper bound $c = s'(n)$ of the space complexity of P_n, which yields the upper bound of maximal identifier of a vertex. As assumed, this can be computed in space $O(s(n))$ and in time $O(p_1(s(n)))$. The time elapsed is constant for a given n. When we know the number of vertices, we can also compute the number of families of every monochromatic graph, since the number of families in every subgraph $G^M_{\sigma_i}$ is constant and the number of subgraphs in the global graph is fully determined by its size, which is known. We have collected (and written to a working tape) all the header data of the encoding of P_n. It takes space $O(\#\Sigma \cdot s(n)) = O(s(n))$ and time $O(p_2(s(n)))$. We compare b with the size of the header. If it is not larger, we return the desired bit and stop.

We now know the size of a record (since the identifiers are stored in binary form, it suffices to use $O(c)$ bits for both vertex and family identifiers) and the lengths of the arrays have also been computed. We perform some arithmetics on b (which is also an $O(c)$-bit number) and obtain the number of array we are asked for and an index in it.

If we are asked for a decision variable of a vertex, we extract the position of the input tape head of the QTM from the vertex identifier using simple arithmetics (it is stored in some intermediate bits of the identifier) and return the desired bit of the position. However if we are asked for a variable the computation decides on in the reverse direction, we have to correct the obtained position by the input tape head movement in the reverse step. This movement is resolved by a lookup to a (constant) table hard-coded into the construction machine, once we know the internal state of the QTM. This state is also extracted from the vertex identifier using simple arithmetics.

If we are asked for a transition type of a family, we decompose the family number into a pair [*subgraph number, identifier of the family in the subgraph*]. The subgraph number can be ignored and the transition type of the family in the subgraph is resolved by a lookup to a (constant) hard-coded table.

If we are asked for a translation table between a vertex number and a pair [*family identifier, index in the family*], we progress analogously. Both vertices and families are indexed in a systematic way according to the subgraph pattern, we just perform some simple arithmetics, lookups to (constant) hard-coded tables, and check some side conditions. Let us just state that for computing the translation of a family child, we need to take into account shifts of the tape heads determined by this child — these tables are also hard-coded. The method is valid both for forth and back translation tables.

We conclude that all computational steps together use no more than $O(s(n))$ temporary space and $O(p(s(n)))$ time. Let us show that such TM can be made reversible. There are two ways of doing this: either we program the machine very carefully to preserve reversibility (which is not difficult in this case) or we program just some TM and convert it into a reversible one by a general algorithm. This algorithm is not described here, but it is well known.

The construction machine returns the result in its internal state and it has to clean the work tapes. This can be achieved by a so-called *uncomputation.*[2] The computation is run in the forward direction, the result is stored

[2]uncomputation is running a computation in the reverse direction

in the internal state, and then the computation is run in the backward direction. During the uncomputation stage, the work tapes and everything is gradually uncomputed into the initial state.

It remains to show that the length of computation can be made constant. Every block of computation can readily be done obliviously, but the computation takes distinct time for distinct blocks of the encoding. It does not really matter for the simulation (see Theorem 5.3 for a particular algorithm), however if we wish to fulfil exactly the requirements of the construction machine we use the following trick. The lengths of computation in individual blocks can be expressed by a formulae. We compute the maximum of the lengths among distinct blocks and justify faster computations to this maximal length by inserting delay loops at the end of the computation. □

Note 5.2 Notice that the construction machine of the sequence of QN's obtained by converting a QTM has low both the space and time complexity. We see that if a problem is solvable by a QTM, it is also solvable by a uniform sequence of QN's with low-complexity construction machine, and vice versa (thanks to Theorem 5.4).

Hence if Definition 3.3 of quantum complexity classes on page 31 (using QTM's) was reformulated using uniform sequences of QN's with low-complexity construction machine, nothing would change. The new definition would be similar to Definition 2.11 of classical complexity classes on page 13 (using BP's).

Theorem 5.2 Every QTM M with advice with bounded space complexity can be simulated by a sequence of QN's P with finite number of transition types. The space and time complexities are preserved by the simulation.

Proof. This theorem is yet simpler to prove. The QTM M for a fixed input size n will be transformed into a QN P_n in the same way. Since we need not prove the uniformness of P, the space restriction of M needs not be computable. Nevertheless the first part of the proof still holds, thus P has a finite number of transition types.

However the QTM M with advice uses one more working tape. Fortunately it does not complicate anything. The fact that M can use a special instruction ask-for-advice is also easy to simulate: every possible answer corresponds to a particular edge inserted into the target network P_n. We do know which one at construction time, but it does not matter, since we shall just prove the existence of P_n and need not construct it explicitly. □

5.2 Converting a sequence of QN's to a QTM

This section is concentrated on the proof of the following theorem.

Theorem 5.3 Every sequence of QN's P with finite number of transition types can be simulated by a QTM M with advice operating in the same space and with time overhead polynomial in the space complexity and linear in the input size. The advice of M is equal to the encoding of P_n.

Proof. We shall construct the simulation QTM M. We must pay attention on that M must be oblivious, since the interference pattern of P_n has to be preserved. The fact, that M has to be reversible, is another complication. The solution provided fulfils both constraints.

The input alphabet Σ of M is equal to the input alphabet of P. Let us choose two distinct letters from Σ and denote them by $0, 1$. They will be used for work tapes. The output alphabet of M is also equal to the output alphabet of P. The target QTM has the following (read-write) work tapes:

1. it contains the identifier of the current state of the simulated quantum network P_n,
2. it contains the number of the input variable P_n currently decides on,
3. it contains the identifier of the current family (in the monochromatic graph with colour equal to the value of the desired input variable) and the index of the vertex in this family,
4. it is a temporary tape for completing records from the bits collected by the inquiries on the advice.

The internal state of the QTM has the form [*computational step, temporary data storage for tape communication, colour corresponding to the value of the input variable, transition type, index of the current vertex in family with given transition type*]. First three items are accessed during all the computation, the last two items serve for the part implementing the actual quantum branching.

The simulation proceeds in the following way. It is described as a sequence of computational steps, however it is clear how to translate it into the language of Turing Machines.

Initialise. A header of the simulated QN P_n is read from the advice, the identifier of the starting state is written to the first work tape. Everything must be done in reversible way, it suffices to use the XOR operator[3]

[3] XOR is an exclusive OR: $x \oplus y = (x + y) \bmod 2$.

as the write operation, since the work tapes are clean in the beginning. Remember that the temporary work tape must be cleaned after the computation, e.g. by uncomputation. This cleaning process is involved in every computational step.

Measure. We measure (destructively) the output tape. If there is something written there, the computation is interrupted and a corresponding result is returned. Otherwise a loop comprising of all following steps is launched. The measurement is then performed again. This step violates the quantum reversibility, but it is the only one.

Do 1. The index i of the input variable x_i the computation decides on in the forward step is extracted from the advice. This process is described in detail later. The index i is written to the second work tape. A complementary number $n-i$ of spare input variables on the right side of the desired one is computed and also written to the second work tape.

Do 2. The input tape head, currently in the beginning, is shifted to the desired place. The value of the input variable (i.e. the corresponding colour) is read into the internal state. The input tape head is shifted right again to the place after the last input variable. This ensures the constant length of the computation. How the shifts are performed, is also described in detail later.

Do 3. We now know the value of the input variable and we have chosen the corresponding monochromatic graph. We translate the identifier of the current state into a family identifier and an index of the vertex in the family (current state being a parent). The result is written to the third work tape. Again, see the following text for the details.

Do 4. We clear the first work tape containing the identifier of the current state. We indeed perform a reverse lookup on the family identifier and the index of the vertex in the family using a reverse translation table. The result (identifier of the state) is XOR-ed to the work tape, which clears it. Again, see the following text for the details.

Do 5. We extract the family transition type from the advice. This type is immediately read into the internal state (since it belongs to a fixed set).

Do 6. We read the index of the vertex in the family into the internal state. This can be done since the index also belongs to a fixed set. After we

have completed the reading, we clear the index on the work tape in second pass (in can be done in reversible way, since we still hold it in the internal state and thus we can perform the XOR operation).

Quantum. We perform the quantum branching — the QTM is in proper internal state and we evolve the state according to the appropriate unitary transformation. Only the fifth item of the internal state (index of the current vertex in family) is changed by this evolution and the tape heads are unchanged. Henceforth, the index of the vertex in the family means the index of the child instead of the parent. This is the only proper quantum step of the algorithm, all other steps are just reversible.

Undo 6. Now we shall do the converse of all previous steps of the loop: we begin by writing the index of the vertex in the family from the internal state to the work tape. After we have completed the writing, we clear the item in the internal state by another reading pass.

Undo 5. We clear the family transition type in the internal state by uncomputing the step *Do 5.*

Undo 4. We translate the family identifier and the index of the vertex in the family to the identifier of the current state by a reverse lookup similar to the one performed in step *Do 4.* Be careful to use a proper translation table, since the index in the family means the index of the child instead of the parent now.

Undo 3. We clear the second work tape (containing the family information) by a lookup similar to the one performed in step *Do 3.* The information read from the advice is XOR-ed to the work tape, this performs indeed the erasure of the information. Again, be careful to use a proper translation table.

Undo 2. We clear the colour information in the internal state by uncomputing the step *Do 2.* We shift the input tape back to the position, where the input variable was read. We read the square again to clear the item and shift the input tape back to the beginning.

Undo 1. We clear the complementary number $n - i$ by an uncomputation. The index i of the input variable x_i the computation has decided on in the previous step (in other words decides on in reverse direction now) is cleared using the well-known trick again. The desired information is read from the advice and XOR-ed to the work tape. Be careful to

read the proper record, since we now ask for the reverse direction of computation.

It is straightforward to see that this algorithm indeed simulates exactly the behaviour of the QN P_n and that the whole simulation is reversible and oblivious. It remains to comment in detail individual steps of computation and to show that the same holds also for them.

Extracting records from the advice: Since the QN P_n is encoded in a highly regular way (data structure sizes are read from the header and the records are stored in arrays of known lengths), we need not perform a sophisticated searching. We just calculate the indexes of the desired bits, it involves a simple arithmetics that can be done in reversible and oblivious way. We then either copy the desired bits into another work tape or read them directly into the internal state.

Even if the records of the advice were stored in sorted lists, we would still be able to perform fast lookups by a binary search (be careful to implement it in an oblivious way and thus not optimise it by interrupting the search when the record is prematurely found). However this would increase the space complexity, since the intermediate indexes of the binary search would have to be remembered for the sake of the reversibility (clearing the temporary space would be done in another pass by the uncomputation).

There is an interesting subproblem in searching — comparing the c-bit numbers. We first compute their difference. To check, whether the result is nonzero, we need to perform a logical disjunction of c bits. For the sake of reversibility, it can not be done in one pass. We can use an approach of halving iteratively the number of bits to disjunct until we obtain a constant number of them. If we know that the result is nonzero, its sign can be checked by simply reading its highest bit. When we know the result of the comparison, we can control another quantum operation by it, e.g. computing the bounds of the left half of the search space is controlled by the condition result < 0 and vice versa.

If the lists were not stored sorted, the search performed would have to be exhaustive, which is pretty slow since the length of the encoding is exponential in the space complexity of P_n. However the space complexity would be lower in this case, since after the comparison is performed and copying of the target bits controlled by the condition result $= 0$ is performed, the temporary results can be immediately uncomputed and the wasted space freed. Be careful to not stop the search when the result is found, since we have to ensure the oblivion.

Shifting the input tape: We are given a number k of squares to shift the tape head to the right. It is written on a work tape. Let us append a counter i to the work tape (of the same size as k, i.e. it is a $\lceil \log_2 k \rceil$-bit number). It is initialised to zero. A loop with the input/output condition[4] $i = 0$ is executed. The body of the loop comprises of shifting the tape head and incrementing i modulo k. The increment operation is reversible and it can be done in an oblivious way, hence the loop in global is also reversible. It is not oblivious though, however if we concatenate two such loops with shifts $k_1 + k_2 = n$, the joint operation is oblivious.

Let us investigate the space overhead of the simulation. We assume the space complexity of P is $\Omega(\log n)$ to be able to fit the position of the input tape head into it. The first and the third work tapes have the length linear in the space complexity of the problem. The second work tape has the length $O(\log n)$, since there are 4 indexes of the input tape head stored there. The length of the fourth work tape depends on the chosen search algorithm. Since the encoding of P_n is stored using arrays, we need to be able to perform some arithmetics, hence it suffices to have $O(s(n))$ space available. The same holds in the exhaustive search case, however we would need to store $\Theta(s^2(n))$ bits there in the binary search case.

The time overhead of the simulation is equal to the number of computational steps performed in the loop. To extract the advice bits, it takes time $O(p_1(s(n)))$ to perform the arithmetics and time $O(\max(\log n, s(n)))$ to copy the results. If the binary search is used, it would take time $O(p_3(s(n)))$, the exhaustive search would take time $O\left(p_4(s(n)) \cdot 2^{s(n)}\right)$. The shift of the input tape takes time $O(n)$. We conclude that the simulation can be performed in linear space and that the time simulation overhead is polynomial in the space complexity and linear in the input size. □

Theorem 5.4 Every uniform sequence of QN's P with a construction machine C can be simulated by a QTM M. The space complexity of M is the sum of the space complexities of P and C. The time complexity of M is the product of the time complexities of P and C, and the sum of a polynomial of the space complexity of P plus the input problem size.

Proof. We use the same simulation algorithm as in the previous proof. Every time the simulation has asked for advice, we launch the construction machine instead now. This causes the additional claim for the space and time resources. □

---

[4]we live in a reversible world, thus the condition of a loop serves for controlling both entering and leaving the loop

Chapter 6

Converting QN's to special forms

In this chapter, we shall show that every QN can be converted into an equivalent network with more regular structure at low cost — with constant space overhead and with time overhead linear in the input size and linear in the length of the computation. We investigate the layered, oblivious, bounded degree, and connected layouts.

Both nonuniform and uniform sequences of QN's can be converted. Construction machines of target uniform sequences are also provided.

It happens that some layouts are hard to construct, i.e. their construction machine has either a big space complexity or a big time complexity. If the construction machine is regarded just as a proof of uniformness, it does not matter. It would matter if we simulate the target QN by a QTM. However there is no reason to simulate the target QN since we can readily simulate the equivalent source QN.

6.1 Layered layout

Theorem 6.1 Every QN P_n can be converted into a layered QN P'_n with constant space overhead (adding just one qbit) and with constant time overhead (doubling the length of computation).

The same holds for sequences of QN's. When converting a uniform sequence P constructed by a machine C, the complexities of the construction machine C' of P' are asymptotically equivalent to the complexities of C.

Proof. We can obtain a layered QN P'_n from a raw QN P_n by simply doubling the vertices: every vertex $q \in Q$ has two representatives q_0, q_1 connected by an edge (q_0, q_1) and every original edge (q, q') is replaced by a 'back' edge (q_1, q'_0). It is obvious that the target QN has 2 layers: it per-

forms the identity operation in the first one and the original transformation in the second one (it goes from the last layer to the first one). The input vertices will be the representatives of the original input vertices in the first layer and vice versa.

From the simplicity of the construction of P'_n, it is obvious that the construction machine of P' just calls the construction machine of P, performs a few corrections, and handles some exceptions (renumbering the vertices, considering new families of the first identity layer). □

Note 6.1 We see that QN's layered in this way are indeed not interesting. To get more insight from the layered layout of a QN, we would have to restrict, for example, the number of qbits passed from one pass to the next one, which is an elusive goal depending on the particular task.

We have to use another trick for QBP's, because the graph of a QBP must be acyclic. Since we can not use the back edges going to the first layer, we build the target QBP from slices corresponding to the source QBP.

Theorem 6.2 Every QBP P can be converted into a layered QBP P' with constant space overhead (less than three) and with time overhead linear in the time complexity (since every computational path will have equal length). The same holds for sequences of QN's.

Proof. Let l be the length of the longest computation of the QBP P. To make P layered, we situate l copies of the modified graph G into consecutive order. We then add several constant edges (independent on the input variables) connecting the i-th layer with the $(i+1)$-th one. We have to incorporate the individual input and output vertices into the computation, thus we connect them by chains of temporary vertices to the input and output layers, see Figure 6.1.

The modified graph G is constructed from the graph of the source QN P in the following way: At first, the vertices are doubled, a vertex q is replaced by two representatives q_0, q_1. Every edge (q, q') is replaced by an edge (q_0, q'_1). We then independently reorder the vertices in both layers, the goal is to transpose the input and output vertices to the right, as seen in Figure 6.1.

The correctness of the simulation is straightforward. The original computational steps are just interwoven by identity transformations. The final measurement of shorter computational paths is deferred to the last layer. The vertices at the ends of the chains are labelled in the same way as the output vertices of G.

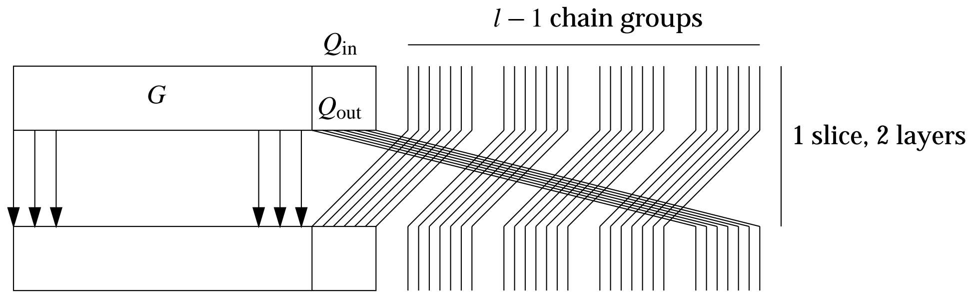


Figure 6.1: Constructing a layered QBP

Let us evaluate the space complexity — the target graph has $2l \cdot (\#Q + (l-1) * \#Q_{\text{out}})$ vertices. Since both $l, \#Q_{\text{out}} \leq \#Q =$ size of P, the target graph has $\mathrm{O}\left((\#P)^3\right)$ vertices and the space overhead is less than three. The computational paths of the target QN have constant length equal to the double of the length of the longest computational path of P. Since the shortest computational path is at least 1 computational step long, the time overhead is not bigger than the time complexity itself. □

Theorem 6.3 Every uniform sequence of QBP's P constructed by a machine C can be converted into a layered uniform sequence of QBP's P' with the overhead mentioned in Theorem 6.2.

There exists a construction machine C'_1 of the target sequence P' having quadratical space complexity (however having super-exponential time complexity) and a construction machine C'_2 having time complexity polynomial in the size of P_n (however having exponential space complexity).

Proof. The RTM C' has to compute the encoding of P'_n and it is allowed to ask for the encoding of P_n as an advice.

The layout of P'_n is fully determined if we know the following numbers: the size of P_n, the number of output vertices of P_n, and the length of the longest computation of P_n. The first one can be read from the header. The second one can be computed by examining all vertices of P_n consecutively and counting them. Computing the third one is a crux — we can either overestimate it by the size of P_n (and thus attenuate the efficiency of the simulation) or compute it exactly by exploring the graph of P_n.

It turns out that computing the diameter of the graph needs either a big space or a big time. We can translate this problem to the so-called *Reachability problem* in this way: The question whether the diameter is smaller than a given threshold t can be solved by determining the number c_1 of

pairs of connected vertices with the distance shorter than t and comparing it with the number c_2 of pairs of connected vertices. The answer is yes iff $c_1 = c_2$. If we can solve the previous question, the diameter can be localised by the binary search algorithm over $t \in \{0, 1, \dots, \#P_n\}$. For exploring the complexity of the involved Reachability problem, look at Lemma 6.4.

If we know the layout of P_n', most of the encoding of P_n' can be computed rather easily — all additional vertices, edges, and thus also families are bestowed regularly along the layers. The families of G^n are also inferred from their counterparts of P_n. However, when computing the translation tables, we have to take into account the renumbering of both layers of G^n (the transposition of the input and output vertices to the right). Both directions of the transposition can be computed by simply counting over vertices of P_n (e.g. the target position of a transposed input vertex of G^n is a constant inferred from the layout plus the number of input vertices with smaller index).

Hence computing a bit of the encoding involves the computation of the layout constants, computing which record are we asked for, performing some arithmetics on the vertex/family identifier to find out the block of the graph, and the evaluation of the inquiry. Recall that we have to be careful to justify the computation length to the longest one.

Let $s(n) = \lceil \log_2 \#P_n \rceil$ be the space of complexity of P. Let us neglect the space and time complexities spent in solving the Reachability problem (with parameters $v = t = 2^{s(n)}$) since it is a dominating term. The space and time complexities of C are also not counted here. When we look over the algorithm, wee see that the space complexity of C' is $O(s(n))$ and the time complexity is $O\left(p(s(n)) \cdot 2^{s(n)}\right)$.

If we merge the results, we conclude that we can provide a construction machine working either in space $O\left(2^{s(n)} \cdot s(n)\right)$ and in time $2^{O(s(n))}$ or in space $O\left(s^2(n)\right)$ and in time $2^{s^2(n)+O(s(n))}$. □

Note 6.2 Both vertices and families of the target layered QBP are numbered consecutively along the layers from the top to the bottom, the vertices in a layer are even numbered consecutively from the left to the right. It now seems just as a byproduct, but the regular numbering of vertices will turn out to be a nice simplification for the oblivious layout.

At this moment, we can readily extend the header of the encoding of a layered QBP by adding the number of layers and the number of vertices in a layer. The construction machine needs to compute them anyway, hence it does not complicate it at all. However it will also simplify the oblivious

layout in the next section. Both matters can be readily managed also for layered QN's.

Lemma 6.4 It is possible to solve the *Reachability problem* (we are given an oriented graph G having v vertices, two vertices v_1, v_2, and a threshold t; the task is to determine whether there is a path from v_1 to v_2 in G shorter than t) using the following resources. They are expressed both in pure variables t, v and in the special case $t = v = 2^s$.

i. space $\mathrm{O}(v \cdot \log t) = \mathrm{O}(2^s \cdot s)$ and time $\mathrm{O}(p(v)) = 2^{\mathrm{O}(s)}$,

ii. space $\mathrm{O}(\log v \cdot \log t) = \mathrm{O}\left(s^2\right)$ and time $\mathrm{O}\left((2v)^{\log_2 t} \cdot p(v)\right) = 2^{s^2 + \mathrm{O}(s)}$.

Proof. The first algorithm is a classical search of the width. We need to remember a current distance $\in \{0, 1, \ldots, t\} \cup \infty$ for every vertex. We also remember a queue of the vertices being processed. The time complexity is obviously polynomial in v.

The second algorithm is the one used in the proof of the *Savitch's theorem*, see [Sav70]. To answer the task, a recursive approach is used. If $t = 0$ or $t = 1$, then a direct evaluation is performed. Otherwise we check recursively whether there exists a vertex v_m such that there exist both a path from v_1 to v_m of length $\leq \lceil t/2 \rceil$ and a path from v_m to v_2 of length $\leq \lfloor t/2 \rfloor$. This involves $2v$ recursive calls at every level. There are $\log_2 t$ levels. □

Note 6.3 This algorithm can be also effectively used for converting a QN into a QBP. Having a promise that every computation finishes indeed in l computational steps, we can build up the target QBP from l slices comprising of the source QN. However we have to ensure, in addition, that the graph of the target QBP is connected — one of the following sections concerns that. (There is also another requirement of a QBP prescribing that the starting vertex must be an input vertex. However everything would work even with this requirement omitted.)

This method can be used, for example, to obtain a QBP from the QN obtained after the reversibility is achieved (by the back-tracking algorithm) on a given deterministic BP.

6.2 Oblivious layout

Let us suppose P_n is a layered QN. We want to convert P_n into an oblivious form, i.e. P'_n will be not only layered but it will also decide on exactly one

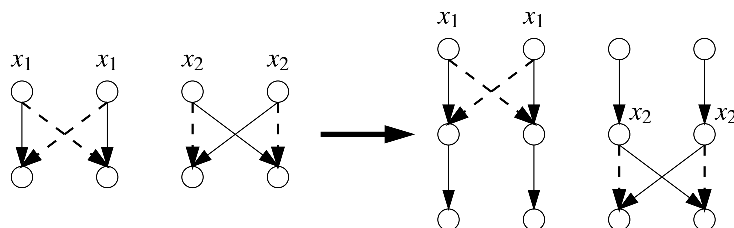


Figure 6.2: Converting a layered QN to an oblivious QN

input variable in every layer. We shall show that such conversion exists and that it is effective.

To convert a uniform sequence of QN's, we have to provide a construction machine C' of the target oblivious sequence P'. To simplify the task maximally, we suppose the encoding of a layered QN/QBP has been extended in the way described in Note 6.2 — the number of layers and the number of vertices in a layer are provided in addition and the vertices are guaranteed to be numbered consecutively along the layers.

Theorem 6.5 Every layered QN P_n can be converted into an oblivious QN P'_n with constant space overhead (less than two) and with time overhead linear in the input size.

The same holds for sequences of QN's. When converting a uniform sequence P constructed by a machine C, the complexities of the construction machine C' of P' are asymptotically equivalent to the complexities of C.

Proof. We divide every layer into n sub-layers, the vertices in the i-th sub-layer decide on the input variable x_i. Every family from the original layer is situated into the sub-layer corresponding to its decision variable and the identity operation is performed in other sub-layers. See Figure 6.2 for an outline.

The length of the computation is increased n times, where n is the input size. The number of vertices is increased by the same factor. If we assume the QN looks at every input variable, then the number of vertices is bigger than the input size and thus the space complexity is less than doubled.

It remains to implement the construction machine C' of the target uniform sequence P'. Let us first decompose the bit number b into a block number and a record number in that block.

- the header can be computed very easily from the source header, all numbers can be obtained by an arithmetic expression, remind that

we also provide the extended header,

- the decision variable of a vertex is extremely simple to determine, since we just need to arithmetically compute the sub-layer number,
- the families are numbered in the way that the identifiers of the original families are left untouched (we allocate indexes $0, 1, \dots, f$ for them) and they are followed by a plenty of one-edge families performing the identity operations — hence computing the transition type of a family is a simple task; these one-edge families are indexed regularly by the triple [original layer number, column number, sub-layer identifier],
- the translation tables must handle the peculiarity of the family numbering: an one-edge family identifier is translated into a target layer number by the expression

$$\textit{layer number} \cdot \textit{input size} + \textit{sub-layer identifier} + c,$$

where $c \in \{0, 1\}$ is a correction depending on whether the original family of the corresponding original vertex is bestowed below or above the sub-layer of the desired one-edge family. In Figure 6.2, it holds that $c = 1$ for the two left one-edge families and $c = 0$ for the two right ones.

When we translate a family number to a vertex identifier, we check whether the asked family is an original one or a new one-edge one. The first one is easy to compute, we obtain the vertex identifier from the advice, ask for its decision variable, and perform some final arithmetics. The second one is also simple, we just compute the correction c by asking for the decision variable of the corresponding original vertex and perform some final arithmetics.

The reverse translation tables are computed similarly. We decompose the vertex identifier, ask for the decision variable of the corresponding original vertex, and see immediately whether it belongs to an original family or to a new one-edge one. We also obtain the correction c by this method.

All assertions in the proof hold both for a QBP and for a QN (with back edges going from the last layer to the first one), however these back-edges have to be handled. Remember that the pure code needs to be wrapped by a quantum wrapper: it must be reversible with constant length of computation and it must clean its work tapes.

We conclude that every bit of the target encoding can be computed by a few inquiries for the advice and some simple arithmetics. Let us neglect the complexities of the source construction machine C. The construction machine C' has the space complexity $\mathrm{O}(s(n))$ and the time complexity $\mathrm{O}(p(s(n)))$. □

Note 6.4 The introduced conversion is worthy for its simplicity, however it is not optimal in most cases. It could happen that only a few input variables are present in the labels of vertices from a layer. It is a waste of time to expand every layer into all n sub-layers.

Hence we can conclude that QN's can be converted in a better way (by omitting sub-layers containing only identity edges). Nonuniform sequences need not be handled in a special way. However the implementation of the cleverer construction machine C_2' of the target uniform sequence would be slightly more complicated: for computing the coordinates of a vertex, we would have to construct and explore all the graph above, because there is no implicit way of obtaining the number of sub-layers the layers above have expanded to. It could be done by a few simple loops, however the complexities of C_2' would be increased.

6.3 Bounded degree layout

Let P_n be a QN. The task is to convert P_n into a d-bounded degree QN P_n', i.e. into a QN with both the input and output degree of the vertices bounded by d. We shall show that such conversion is always possible.

Theorem 6.6 Every QN P_n can be converted into a 2-bounded degree QN P_n' with constant space overhead (just adding a few qbits) and with constant time overhead (depending on the maximal family size).

The same holds for sequences of QN's. When converting a uniform sequence P constructed by a machine C, the complexities of the construction machine C' of P' are asymptotically equivalent to the complexities of C.

To we prove the theorem, we need to claim some intermediate lemmas.

Definition 6.1 We say that an operator $M : \ell_2(C) \to \ell_2(C)$ is a ***rotation of vectors*** $i, j \in C$ ***by angle*** α iff $i \neq j$ and

$$M = I_C + (|i\rangle\langle i| + |j\rangle\langle j|)(\cos\alpha - 1) + (|j\rangle\langle i| - |i\rangle\langle j|)\sin\alpha,$$

where I_C is the identity operator on $\ell_2(C)$ and we denote it by $M = R_{i,j}^C(\alpha)$. The superscript C can be omitted if it is clear which space is meant from

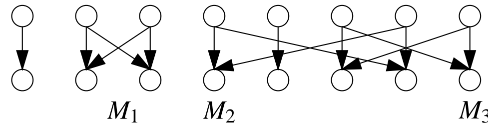


Figure 6.3: Representation of an independent set of 3 rotations by a QBP

the context. We say that a set of rotations $S = \{M_k\}_{k \in K}$, $M_k = R^C_{i_k,j_k}(\alpha_k)$ is ***independent*** iff

$$\#\left(\bigcup_{k \in K} \{i_k, j_k\}\right) = 2 \cdot \#K,$$

i.e. the indexes are pairwise distinct.

Example 6.1 A rotation of vectors $2, 0$ on space $\ell_2(\{0,1,2\})$ by angle α can be expressed in the computational basis as

$$R_{2,0}(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}.$$

Lemma 6.7 Every independent set $S = \{M_i\}_{i=1}^k$ of rotations operating on a space $\ell_2(C)$ can be performed by a layered 2-bounded degree QBP having just two layers. Vertices in a layer correspond to vectors $c \in C$.

Proof. Let us take the product of $M = M_1 \cdot M_2 \cdots M_k$ of the rotation operators and represent it by a bipartite graph B with labelled edges. From the independence of S we see that M has either one or two nonzero items in every row and in every column. Moreover under an assumption that no rotation angle is a multiple of $\pi/2$, it holds that the family decomposition of B yields k families of size 2 and $(\#C - 2k)$ one-edge families performing the identity operation. If there are some bad angles, some size-2 families will punctuate. Look at Figure 6.3 for an example. □

We shall show how to decompose a unitary operator $U : \ell_2(C) \to \ell_2(C)$ into a product of $\Theta(d^2)$ rotations (let $d = \#C$). This immediately implies the ability of representing U by a layered 2-bounded degree QBP having $\Theta(d^2)$ layers. However if we order the rotations well, it is possible to divide the rotations into $\Theta(d)$ groups of consecutive *independent* rotations. It follows that the QBP can be indeed compressed to $\Theta(d)$ layers.

Lemma 6.8 Every nonsingular operator $M : \ell_2(C) \to \ell_2(C)$ can be decomposed as $M = VT$ where V is a product of $\Theta\left(\#C^2\right)$ rotations (thus it is also a unitary operator) and T is an upper triangular matrix (in the computational basis). The rotations form $\Theta\left(\#C\right)$ consecutive groups of independent rotations. Moreover if M is unitary, then T is necessarily a diagonal unitary matrix.

Proof. Look at the following matrix E. The nonzero items of E denote the order in which the items of M are consecutively eliminated. The zero items are henceforth ignored. If the size of M is $n \times n$, then $e = n(n+1)/2 = \Theta\left(n^2\right)$. We see that the sequence of eliminations consists of $2n-1 = \Theta(n)$ consecutive diagonals and the eliminations are independent on each diagonal.

$$E = \begin{pmatrix} 1 & 0 & & & & \\ 2 & 3 & 0 & & \ddots & \\ 4 & 5 & 7 & 0 & & \\ 6 & 8 & 10 & 13 & 0 & \\ 9 & 11 & \ddots & \vdots & e-3 & 0 \\ 12 & & \dots & e-2 & e-1 & e \end{pmatrix}.$$

Let us describe how the elimination procedure works. We have a register M_k containing the modified operator after the k-th step was performed. In the beginning, we set $M_0 = M$. We then consecutively eliminate items in the order specified by the matrix E. The k-th step eliminating the item $[i, j]$ comprises of the following operations:

- If $i = j$, i.e. if we are processing a diagonal element, ensure that $\langle j|M_k|j\rangle \neq 0$. If $\langle j|M_{k-1}|j\rangle$ is zero, find a non-zero item in the same column, produce a row permutation operator P, and update $M_k = PM_{k-1}$. Otherwise leave $M_k = M_{k-1}$ unchanged. Such nonzero item certainly exists, since M and thus also all M_l's are nonsingular operators.

- If $i > j$, we produce a rotation operator $U_{i,j} = R_{i,j}(\alpha_{i,j})$, where $\alpha_{i,j}$ is chosen such that $\langle i|U_{i,j}M_{k-1}|j\rangle = 0$. Let $M_k = U_{i,j}M_{k-1}$. The appropriate $\alpha_{i,j}$ always exists and moreover the property that $\langle j|M_k|j\rangle \neq 0$ will be preserved:

  (Let $a = \langle j|M_{k-1}|j\rangle$ and $b = \langle i|M_{k-1}|j\rangle$ be the items of the source matrix M_{k-1} and let $a' = \langle j|M_k|j\rangle$ and $b' = \langle i|M_k|j\rangle$ be the items of the

resulting matrix M_k.

$$\begin{aligned}(b' =) \quad 0 &= -\sin\alpha \cdot a + \cos\alpha \cdot b \\ \tan\alpha &= b/a \neq \infty \qquad \text{(since } a \neq 0\text{)} \\ a' &= \cos\alpha \cdot a + \sin\alpha \cdot b \\ &= \cos\alpha \cdot (a + \tan\alpha \cdot b) \\ &= \cos\alpha \cdot (a + b^2/a) \neq 0,\end{aligned}$$

since $\cos\alpha \neq 0$ and $a^2 = -b^2$ has no real solution.)

The rotation operator $U_{i,j}$ only touches items on the i-th and the j-th row. However, from the order of the elimination, we see that all items to the left of $[i, j]$ and $[j, j]$ have already been eliminated to 0 and thus they are left unchanged. We conclude that $U_{i,j}$ leaves zero values in the already eliminated items and that it eliminates the item $[i, j]$ in addition.

After all marked items are eliminated, the modified operator M_e becomes an upper triangular matrix. It holds that $UM = M_e$, hence $M = U^{-1}M_e$. The algorithm finishes and it returns $V = U^{-1}$ and $T = M_e$. We have obtained $2n-3$ groups of consecutive independent rotation operators. The groups correspond to inner diagonals. These groups are occasionally interwoven by permutation matrices.

It remains to show, that if M is unitary, then T is a diagonal unitary operation. We know that T is an upper triangular matrix and that it is also unitary. It follows that T must be a diagonal matrix. □

Corollary 6.9 Every unitary operator $U : \ell_2(C) \to \ell_2(C)$ can be simulated by an equivalent layered 2-bounded degree QBP with $\Theta(\#C)$ layers.

Proof. We decompose U using Lemma 6.8 into $\Theta(\#C)$ groups of independent rotations. Recall that some permutation operators can be interpolated between the groups. We then simulate every such group by a layered 2-bounded degree QBP with 2 layers using Lemma 6.7. We built the target QBP by appending the individual QBP's (as slices). Look at Figure 6.4 for an example representation of a general 6×6 unitary operator that needs not be permuted. The identity edges are omitted for clarity, the first layer performs the phase changes T. □

We have prepared the main tool for simulating complex unitary operation by the simple ones. Let us prove the main theorem.

Proof of Theorem 6.6 Let P_n be a QN. Since P_n is finite, it has also a finite number of transition types. Let us decompose every transition type

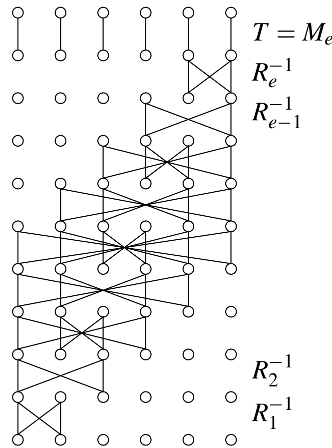


Figure 6.4: Representation of a 6×6 unitary operator (that needs not be permuted) by a QBP

U_i (which is necessarily a unitary operation) using Corollary 6.9 into a 2-bounded degree QBP B_i having l_i layers. Let us compute the maximal number of layers $l = \max_i l_i$. We justify every decomposition B_i of U_i by adding dummy layers to l layers, hence the decompositions of all transition types have equal length.

If we replace every family F having transition type i by the decomposed QBP B_i in the source QN P_n, we obtain an equivalent QN P'_n, because both F and B_i have the same input/output interface and they compute the same function. Moreover the interference pattern is preserved, since all B_i's have equal length of computation (even the simple edges have been stretched to l consecutive edges).

Both the number of vertices and the length of the computation are multiplied by l by this simulation. l is a constant depending on the transition types of P_n. Hence the space complexity is increased by $\Theta(\log l)$ and the time complexity is multiplied by l.

The same argument holds also for sequences of QN's. If P is a nonuniform sequence, we convert every individual P_n separately and the target sequence P' remains nonuniform. If P is a uniform sequence, it has necessarily a finite number t of transition types, thus the parameter l is constant for all P_n's. We convert the source sequence P into the target 2-bounded degree sequence P' in the same way as stated above. Let us show that P' is also a uniform sequence, i.e. there exists a construction machine C' of P'.

We first decompose all t transition types into 2-bounded degree QBP's having l layers and store the description of all decompositions into a (fixed size) table hard-coded into the construction machine C'. Notice that the number of transition types of $\{B_i\}_{i=1}^t$ is also finite, hence the first requirement of the target sequence P' is fulfilled.

Vertices of the target QN P'_n are indexed by the pair [original vertex identifier, sub-layer number k] and families of P'_n are indexed by the triple [original family identifier, sub-layer number k, index j of the family in the sub-layer], where $k, j \in \{0, 1, \dots, l-1\}$. Notice that not all family identifiers are valid.

- the target header can be computed from the source header using simple arithmetics,
- the decision variable of a vertex $q' = [q, k]$ is resolved by asking the advice for the decision variable of the original vertex q (q is extracted from q' using simple arithmetics),
- the transition type of a family $f' = [f, k, j]$ is resolved by asking the advice for the transition type of the original family f and then translating it (using the parameters k, j) using the hard-coded description of the decompositions,
- finally, the translation tables are inferred in a similar way: To translate a family identifier f' and a number i' of a vertex in the family into a vertex identifier q', we decompose $f' = [f, k, j]$, combine k, j, i' using the hard-coded tables into i, ask the advice for the translation of f, i into q, and combine $[q, k] = q'$. This method is valid both for parents and for children.

  To translate a vertex q' into a family identifier f' and a number i' of a vertex in the family, we decompose $q' = [q, k]$, ask the advice for the translation of q into f, i, combine k, i using the hard-coded tables into j, i', and combine $[f, k, j] = f'$. This method is also valid both for parents and for children.

Remember that the pure code needs to be wrapped by a quantum wrapper: it must be reversible with constant length of computation and it must clean its work tapes.

We conclude that every bit of the target encoding can be computed by a few inquiries for the advice and some arithmetics. Let us neglect the complexities of the source construction machine C. The target construction machine C' has the space complexity $\mathrm{O}(s(n))$ and the time complexity $\mathrm{O}(p(s(n)))$. □

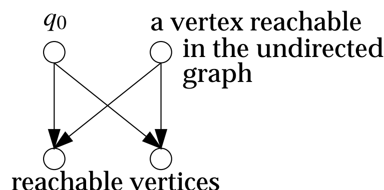


Figure 6.5: Reachable vertices in the QBP of the Hadamard operation

Note 6.5 It is not clear whether a similar regular construction exists for other degree bounds d, however it is not important anyway since the construction for the special case $d = 2$ is already optimal in the following sense. Let M be an $n \times n$ operator and let B be its decomposition.

i. B has asymptotically minimal number of edges: there are $\Theta(n)$ layers, each with n vertices and the degree of a vertex is bounded. Hence B has $\Theta\left(n^2\right)$ edges. A unitary operator of such dimension has $\Omega\left(n^2\right)$ degrees of freedom.

ii. B has asymptotically minimal number of layers: every layer is incident with only $\Theta(n)$ edges, since the number of vertices in a layer is n and the degree of a vertex is bounded. For a general operator M, there are $\Omega\left(n^2\right)$ edges needed, hence $\Omega(n)$ layers is a lower bound. B has $\Theta(n)$ layers.

6.4 Connected graphs

We often work with a QN comprising of a graph with more components. The components other than the starting one are useless, of course. If we get rid of them, we decrease the space complexity while the time complexity is preserved.

We have to be cautious. It might seem that deleting all vertices that are not reachable from the starting vertex is the best way of doing this. However many vertices are essential for the reversibility of the evolution — remind the simple Hadamard operation in Figure 6.5. Deleting vertices that are unreachable in the corresponding undirected graph is a safe way of cleaning without disturbing the quantum properties.

Theorem 6.10 Every QN P_n can be converted into a QN P_n' with connected graph. The space complexity is appropriately decreased, the time complexity is preserved.

The same holds for sequences of QN's. When converting a uniform sequence P constructed by a machine C, the target sequence P' is also uniform and there exists a construction machine C'_1 with quadratical space complexity and a construction machine C'_2 with time complexity linear in the size of P_n.

Proof. Let us consider the set of vertices K that need to be included into the target QN P'. We first add the starting vertex q_0. It is obvious that if it happens for a configuration q, that the amplitude of $|q\rangle$ is nonzero during the computation, then there exists an oriented path from q_0 to q in the graph of P_n. Hence if we add all vertices connected to the starting vertex, we handle the forward direction of the computation. Since we live in a reversible world, we must apply the rule also for the reverse direction. We apply it forth and back while new vertices are found. We conclude that the component K of the undirected graph corresponding to P_n containing q_0 is closed under both directions of computation.

Hence the QN P'_n comprising of the subgraph induced by the component K is equivalent to P_n. The time complexity is preserved and the space complexity has possibly been reduced.

It remains to show, how to implement the construction machine C' of the target sequence P' from the construction machine C of the source uniform sequence P. We use the same tricks as those used in the former construction machines: programming just some machine and achieving the reversibility by a general procedure, the uncomputation for cleaning the work tapes, the waiting loops at the end of the computation to justify the length of it, the arithmetics to compute the type and the number of the record we are asked for (after we compute sizes of the data structures). Again, we shall often use the procedure solving the Reachability problem, see Lemma 6.4.

- some records of the header are trivially computed by looking up to the encoding of P_n, the only interesting records are: the number of vertices of P'_n (computed by repeated calling Reachability(q_0, q) for all q's and summing the results), the number of families in the monochromatic graphs (done in the same way over the families of the source graph, we use some vertex of the family obtained from the translation tables as the second argument q of the Reachability call), and the index q'_0 of the starting vertex (it is equal to the number of reachable vertices having an identifier lower than q_0),
- the decision variable of a vertex q' is answered by a simple lookup to the encoding of P_n for the decision variable of a vertex q, where q is

the q'-th reachable vertex in P_n,

- the family transition types are evaluated in the same way,
- the translation tables are evaluated simply by incorporating both directions of translation (e.g. the parents $p'_1, p'_2, \ldots$ of the family f' can be obtained by looking up for parents $p_1, p_2, \ldots$ of the corresponding family f and translating them into $p'_1, p'_2, \ldots$).

The complexity of C is determined by the choice of the algorithm solving the Reachability problem. It is analogous to the complexity of the construction machine of a layered QN: we can provide a construction machine working either in space $\mathrm{O}\left(2^{s(n)} \cdot s(n)\right)$ and in time $2^{\mathrm{O}(s(n))}$ or in space $\mathrm{O}\left(s^2(n)\right)$ and in time $2^{s^2(n)+\mathrm{O}(s(n))}$. □

6.5 Conclusion

Theorem 6.11 Every QN/QBP P_n can be converted into a 2-bounded degree oblivious QN P'_n with connected graph. The space overhead is constant, the time overhead is the product of the time complexity (because all computations are justified to equal length), the input size, and a constant.

The same holds for sequences of QN's. When converting a uniform sequence P constructed by a machine C, the construction machine C' of the target uniform sequence P' will have either a big space complexity or a big time complexity: Let s be the space complexity of P. Then C' works either in space $\mathrm{O}\left(2^{s(n)} \cdot s(n)\right)$ and in time $2^{\mathrm{O}(s(n))}$ or in space $\mathrm{O}\left(s^2(n)\right)$ and in time $2^{2 \cdot s^2(n)+\mathrm{O}(s(n))}$. The complexities of C are not counted here.

Proof. The source QN/QBP P_n is converted consecutively into a layered (Theorems 6.1 and 6.2), an oblivious (Theorem 6.5), a 2-bounded degree (Theorem 6.6), and a connected (Theorem 6.10) form. Space overheads are summed, time overheads are multiplied.

The construction machine C' is composed from the construction machines C_l, C_o, C_b, C_c corresponding to the individual conversion steps. The space complexities are summed, the time complexities are multiplied (this is just an upper bound for the unrealistic case where every computational step of each construction machine involves a call of the lower construction machine). Since the $\mathrm{O}(f(n))$ notation is used, the complexity formulas look almost unchanged. □

Chapter 7

Achieving reversibility

We have proposed a quantum model of computation — Quantum Networks. Then we have proved its equivalence with Quantum Turing Machines. Let us investigate the relationship of QN's with the classical models of computation — Branching Programs and Probabilistic Branching Programs.

The latter models are not reversible in general. The task of this chapter is to develop the conversion procedures achieving reversibility while preserving the acceptance probabilities. Since these procedures exist, it follows that QN's are at least as powerful as their classical counterparts. There are indeed 3 distinct methods how to accomplish this, each has its own advantages and drawbacks.

It turns out that to convert a classical program to a reversible one, either the space complexity or the time complexity have to be increased.

7.1 The back-tracking method

This method is designed for the conversion of a deterministic Branching Program P to the reversible one P'. The space complexity is preserved by the simulation, however the time complexity is substantially (in worst case exponentially) increased.

It works in the following way: imagine that every edge of a monochromatic graph G_σ of P is doubled — one edge for either direction. Hence the graph G_σ becomes an Euler graph $G_{\sigma,2}$ (with even degrees of vertices). Then the vertices of P' correspond to the edges of $G_{\sigma,2}$. The computational path of P' follows the back-tracking search of the graph $G_{\sigma,2}$ (recall the name of this method — back-tracking method). It is apparent that every vertex has a unique predecessor and a unique subsequent vertex. Look

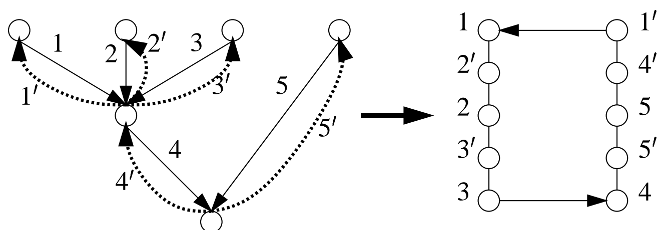


Figure 7.1: Sketch of the back-tracking method

at Figure 7.1 for an example conversion of an individual monochromatic graph.

To complete the conversion of P, we have to combine the converted monochromatic graphs into one global graph, we must be especially cautious to the vertices with input degrees distinct in individual monochromatic graphs. At last we have to punctuate the Euler cycles at appropriate vertices to obtain the input and output vertices of P'.

Theorem 7.1 Every BP P can be converted into a RBP P' with constant space overhead (adding just a few bits) and with time overhead exponential in the space complexity of P. The target program P' does not have an acyclic graph, however every its consistent computational path is acyclic.

Proof. The conversion follows from the sketched algorithm. However we shall describe it using another notation — instead of talking about doubled edges in the individual monochromatic graphs we will index the original vertices by a counter according to the input edge.

Let $P = (n, \Sigma, \Pi, Q^{\text{pc}}, E^{\text{pc}}, q_0, d, v)$ be a BP. Let us modify the graph $(Q^{\text{pc}}, E^{\text{pc}})$ of P by inserting dummy vertices at appropriate places to fulfil the parental condition. (*If there are more than one input variable assigned to the parents of a vertex, we choose one of them x_i and interpolate a temporary vertex deciding also on x_i for every foreign input variable. This is needed since the conversion algorithm needs the source graph having the parental condition already fulfilled. Look at Figure 7.2 for an example. This interpolation is done separately for every monochromatic graph, but the temporary vertices interpolated have its one outgoing edge labelled by every colour.)* The target graph will be denoted by (Q, E). The function returning the decision variable of the parents of a vertex will be denoted by d_p.

Now we will construct the target RBP $P' = (n, \Sigma, \Pi \cup \{\text{err}\}, Q', E', q'_0, d', v')$. Let us denote the monochromatic input degree of a vertex by $D_\sigma(q)$

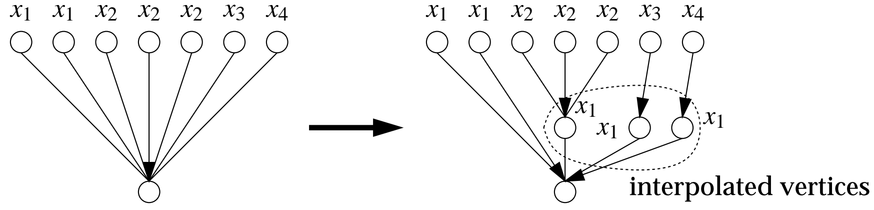


Figure 7.2: Fulfilling of the parental condition in a BP

and the maximal monochromatic input degree of a vertex by $D(q)$:

$$D_\sigma(q) = d^-_{(Q,E_\sigma)}(q), \quad D(q) = \max_{\sigma \in \Sigma} D_\sigma(q).$$

Let us denote the i-th monochromatic parent of vertex q by $\text{parent}_{\sigma,i}(q)$ and the only monochromatic child of vertex q by $\text{child}_\sigma(q)$. The rank of vertex p in the monochromatic list of parents of another vertex q is denoted by $\text{rank}_{\sigma,p}(q)$. The particular ordering is not important, but it must be fixed.

The set Q' of vertices of P' can be decomposed into the following sets:

i. $\{q_{\text{back}} | q \in Q\}$ corresponds to the set of additional backward edges in the sketched algorithm,

ii. $\{q_i | q \in Q \;\&\; 1 \leq i \leq \max(D(q), 1)\}$ corresponds to the set of original forward edges incoming to the vertex (however there are a little bit more of them, since the number of target vertices is constant for all colours). The input vertices have the input degree zero, a new vertex is allocated for them.

We define a shortcut q_{decide}, it denotes the vertex corresponding to the last incoming edge. Notice that if the back-tracking search has arrived into this vertex, it continues by following the forward edge going from the original vertex q.

$$q_{\text{decide}} = q_{\max(D(q),1)}.$$

The next step is to define the edges of P'. We shall construct E', d', and v' in one pass. The starting vertex of P' will be $q'_0 = q_{\text{decide}}$ for $q = q_0$. Let us browse all vertices of the target graph and state for every vertex $q' \in Q'$ its decision variable and all its outgoing edges:

i. q_{back} for $q = q_0$: it is an output vertex denoting that an error has occurred. A valid computational path should never reach this vertex, however it is needed for the reversibility. $d'(q_{\text{back}}) = \text{err}$.

ii. q_{back} for $q \in Q_{\text{inp}}, q \neq q_0$: it revolves the computation back into the forward mode, it decides on $d(q)$ and all its output edges go to q_{decide}.

iii. q_{back} for $q \in Q_{\text{child}} = \overline{Q_{\text{inp}}}$: it keeps back-tracking going up, it decides on $d_p(q)$, the output edge for colour $\sigma \in \Sigma$ is determined in this way:

- if $D_\sigma(q) \geq 1$ then it goes to p_{back}, where $p = \text{parent}_{\sigma,1}(q)$ is the first parent of q,
- if $D_\sigma(q) = 0$ then it goes to q_1 (back-tracking revolved).

iv. q_{decide} for $q \in Q_{\text{out}}$: it is an output vertex labelled by the corresponding result. $d'(q_{\text{decide}}) = d(q)$.

v. q_{decide} for $q \in Q_{\text{par}} = \overline{Q_{\text{out}}}$: it moves forth into the following vertex (being careful on the appropriate rank), it decides on $d(q)$, the output edge for colour $\sigma \in \Sigma$ goes to $c_{\text{rank}_{\sigma,q}(c)}$, where $c = \text{child}_\sigma(q)$ is the only child of q.

vi. q_i for $q_i \neq q_{\text{decide}}$: it keeps on back-tracking since in is not the last incoming edge, it decides on $d_p(q)$, the output edge for colour $\sigma \in \Sigma$ is determined in this way:

- if $i < D_\sigma(q)$ then it goes p_{back}, where $p = \text{parent}_{\sigma,i+1}(q)$ is the next parent of q,
- if $i \geq D_\sigma(q)$ then it goes to the next incoming edge q_{i+1} (skipping the superfluous input edges).

This description may seem rather involved, in fact it is involved. However if we look closer, we see that the target RBP fulfils all requirements:

- it has $1 + \#Q_{\text{out}}$ input vertices (q_{decide} for $q = q_0$ and q_{back} for $q \in Q_{\text{out}}$) and $1 + \#Q_{\text{out}}$ output vertices (q_{back} for $q = q_0$ and q_{decide} for $q \in Q_{\text{out}}$), the starting vertex is the sink of the graph,
- the forward step of computation is well defined in every non-output vertex (exactly one outgoing edge of every colour $\sigma \in \Sigma$),
- the parental condition is fulfilled in the target graph (assuming that it was fulfilled in the source graph),
- thus also the reverse step of computation is well defined in every non-input vertex (exactly one ingoing edge of every colour $\sigma \in \Sigma$),
- the target graph is connected,

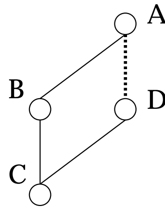


Figure 7.3: Inconsistent cyclic path obtained using the back-tracking method

- well, it is not acyclic, but every consistent computational path is acyclic. For a counterexample that can not be unfortunately remedied look at Figure 7.3: the computation back-tracks consistently via the vertices $A - B - C - D$, it will consistently return to C afterwards, however there ***exists*** a backward edge to A — this edge is necessarily inconsistent with the previous computation because there is only one edge outgoing from A and this is another one — and its existence can not be avoided.

Let us compute the space complexity of P'. The number of vertices interpolated for having parental condition fulfilled is certainly less than the original number of vertices. Since the output degree of a vertex is fixed, it holds that $\#E = \mathrm{O}(\#Q)$. Hence $\#Q' = \mathrm{O}(\#Q)$ and the space complexity of P' is bigger by some constant.

The longest possible computational path of P' goes twice via every edge. The shortest possible computation of P comprises of one step. Since $\#E' = \mathrm{O}(\#Q')$, we conclude that the time overhead of P' is not worse than exponential in the space complexity. □

Example 7.1 Let us demonstrate the conversion algorithm on a BP P computing the logical conjunction $x_1 \mathbin{\&} x_2 \mathbin{\&} x_3$. The first step of conversion (adding dummy vertices to fulfil the parental condition) is sketched in Figure 7.4. The rightmost sketch denotes the numbering of vertices.

The vertices of the target RBP P' are numbered as following: i_j denotes the j-th incoming edge of the original vertex i and i_B denotes the back edge going to the original vertex i. The final RBP P' has $7+7+1+1=16$ vertices and it is outlined in Figure 7.5. Recall that solid edges are labelled by 0 and dashed edges are labelled by 1. The input/output vertices are highlighted, the decision variable of a vertex is marked by the fill style.

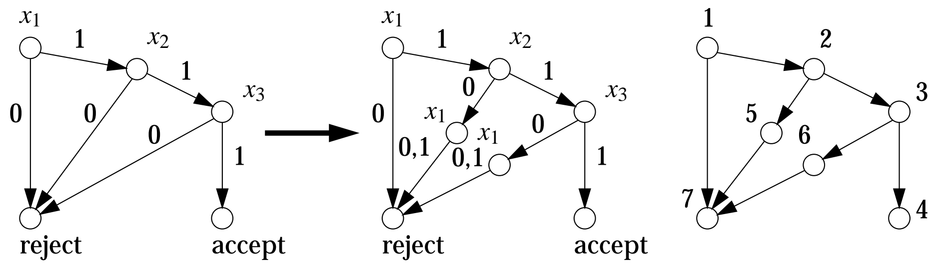


Figure 7.4: Adding dummy vertices to a real BP

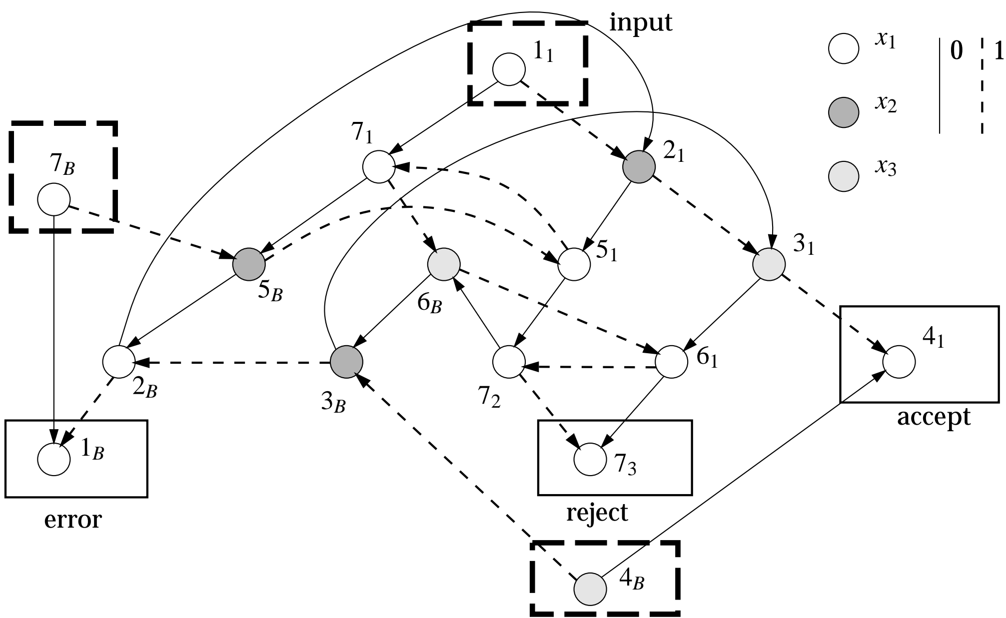


Figure 7.5: Performing the back-tracking on a real BP

Nonuniform sequences of BP's can be obviously converted in the same way. To prove the validity of the conversion method also for uniform sequences we have to provide a construction machine of the target sequence.

Theorem 7.2 Every uniform sequence of BP's P constructed by a machine C can be converted into a uniform sequence of RBP's P' with the overhead and constraints mentioned in Theorem 7.1.

Let s be the space complexity of P. The construction machine C' of P' has space complexity linear in s and time complexity polynomial in s.

Proof. The encoding of a BP has been vaguely defined on page 8. To achieve low complexities of the construction machine, let us modify it slightly in the following way:

i. we add another sorted list of edges — the present list is sorted using source vertices, the augmented list will be sorted using destination vertices,

ii. we split either lists of edges to $\#\Sigma$ smaller lists corresponding to the transitions when $\sigma \in \Sigma$ is read,

iii. we represent the edges by arrays (instead sorted lists), because the binary search would increase the space complexity too much and the exhaustive search would be too slow.[1]

The changes should not make problems to the construction machine C (indeed they simplify it) and they help C' a lot. Notice that the length of the encoding is no more than squared.

Target RBP P'_n will be encoded as a QN according to the Definition 4.9 on page 45. Consider that it is actually no longer a branching program, but a network. However the maximal length of computation is known, hence we can convert the target QN to a QBP using the conversion to the layered form, see Note 6.3 on page 67.

Let us decompose the target construction machine C' into two submachines C'_1, C'_2. The first one C'_1 will interpolate temporary vertices to fulfil the parental condition and the second one C'_2 will perform the actual back-tracking algorithm. The intermediate program P_n^{pc} provided by C'_1 will be encoded in the same way as P_n. The machine C' will be the space preserving conjunction of C'_1 and C'_2, i.e. C'_2 is run and when it asks for a bit of the advice, C'_1 is launched.

[1]this introduces holes in the encoding, moreover there are more backward edges incident with a vertex thus the reverse list needs to be enlarged $\#Q$ times to handle that

Parental condition: The vertices of P_n^{pc} are indexed by the pair [original vertex identifier q, colour $c \in \Sigma \cup \{\mathrm{old}\}$], where $[q, \mathrm{old}]$ corresponds to the original vertex q and $[q, \sigma]$ corresponds to the vertex interpolated after q when input letter σ is read. Not all interpolated vertices are incorporated to the computation, i.e. the numbering contains holes.

- it is clear that the header of P_n^{pc} containing the data sizes can be computed readily from the header of P_n,
- the decision variable of an original vertex $[q, \mathrm{old}]$ is resolved easily by a lookup to the advice, however the decision variable of an interpolated vertex $[q, \sigma]$ is equal to the decision variable of p, where $p = \mathrm{parent}_{\sigma,1}(\mathrm{child}_\sigma(q))$, thus p can be computed by two lookups to the arrays of edges labelled by σ,
- since the produced lists of edges are stored in arrays (rather than sorted lists), we can also decompose the index of the desired bit b into colour σ and vertex identifier $[q, c]$, $c \in \Sigma \cup \{\mathrm{old}\}$ using simple arithmetics.

  When asking for the forward direction of computation (i.e. asking for a record in the array sorted using source vertices), we check whether a vertex has been interpolated after q, i.e. whether $d(q) \neq d(\mathrm{parent}_{\sigma,1}(\mathrm{child}_\sigma(q)))$. If it has, then $[q, \mathrm{old}]$ goes to $[q, \sigma]$ (when σ is read) and $[q, \sigma]$ goes to $\mathrm{child}_\sigma(q)$ (for every colour). Otherwise $[q, \mathrm{old}]$ goes directly to $\mathrm{child}_\sigma(q)$ (when σ is read) and $[q, \sigma]$ is marked as a dummy record (for every colour).

  The reverse direction is computed in similar way. The array of reverse edges is, however, much longer, since it is indexed by the triple [original vertex number q, colour $c \in \Sigma \cup \{\mathrm{old}\}$, index i of the back edge], where $i \in \{1, 2, \dots, \#Q\}$. There are 0 or 1 edges going to an interpolated vertex $[q, \sigma]$, which can be computed as described in the previous paragraph. However because the number of edges going to an original vertex $[q, \mathrm{old}]$ can be quite big, we first identify its original parent vertices $p_1 = \mathrm{parent}_{\sigma,1}(q)$, $p_i = \mathrm{parent}_{\sigma,i}(q)$, which can also be done by a lookup, then we progress as described in the previous paragraph.

Back-tracking: This conversion is slightly simpler than the previous one. The vertices of P'_n are indexed by the pair [original vertex q, type $t \in \{1, 2, \dots, \#Q\} \cup \{\mathrm{back}\}$]. The numbering has holes, since the number of types t really used for a vertex q depends on its input degree, however it

does not matter. A survey along the back-tracking algorithm reveals that the complete behaviour of P'_n is very regular and can be computed using simple arithmetics:

- the header is simple to compute: the maximal identifier of a vertex is known, there are approximately as many families as vertices (minus the number of output vertices), and the starting vertex is also known,
- the decision variable of a vertex $[q,t]$ is either $d(q)$ or $d_p(q)$ depending on whether q is an input vertex in P_n and whether $t < D(q)$, we can obtain both numbers by asking for advice,
- families correspond to their (only) parent vertex, hence every family has decision type 1 (a simple one-edge family),
- both translations between the parent vertex and the family identifier are trivial (identity), the child vertex of the family corresponds to the transition from its parent vertex $[q,t]$: all types of transitions are categorised within the description of the algorithm, so we just compute some of the variables

  $$D(q), D_\sigma(q), \mathrm{child}_\sigma(q), \mathrm{parent}_{\sigma,1}(q), \mathrm{parent}_{\sigma,t+1}(q), \mathrm{rank}_{\sigma,q}(\mathrm{child}_\sigma(q))$$

  from the source graph, which can be done fast since its edges are encoded in arrays, and compose the target vertex from them. The reverse translation corresponds to the reverse transition, which is also regular.

As usually, we wrap both C'_1 and C'_2 by a quantum wrapper (achieving reversibility, cleaning work tapes, and ensuring constant length of computation). The space complexity of C'_1 is $\mathrm{O}(s(n))$. Notice that if the source BP P_n was encoded using sorted lists, it would be $\mathrm{O}(s^2(n))$ instead. The time complexity of C'_1 is $\mathrm{O}(p(s(n)))$. Also the space complexity of C'_2 is $\mathrm{O}(s(n))$ and its time complexity is $\mathrm{O}(p(s(n)))$.

We conclude that even if C'_2 calls C'_1 at every its computational step, the final construction machine C' has total space complexity $\mathrm{O}(s(n))$ and time complexity $\mathrm{O}(p(s(n)))$. We neglect the complexities of the source construction machine C. □

Note 7.1 This conversion works only for branching programs and it does not work for networks. The essential fact the algorithm relies on is that the source graph is acyclic, thus the back-tracking stops in finite time.

Note 7.2 The results of this section imply that both nonuniform and uniform sequences of BP's can be simulated by QTM's in the same space (and exponentially longer time).

7.2 The infinite iteration method

This method serves as a counterpart of the back-tracking method for probabilistic computations. It converts a PBP with random choice type $\{\frac{1}{2}, \frac{1}{2}\}$ to a QN operating in the same space. However the target QN does not halt absolutely, and its expected length of computation is exponentially longer. The method has been proposed by J. Watrous in [Wat98].

The problem of simply replacing the fair coin flip by a quantum operation U (e.g. the Hadamard operation) is the quantum interference. From the unitarity of U some transition amplitudes are negative, which causes that some computational paths interfere with each other. The solution stated avoids these amplitudes by assuming that the argument of the Hadamard operation is reset to $|0\rangle$ every time. The second problem is the irreversibility of transitions. It is circumvented by storing the new state separately and exchanging this new state with the current one. Again we suppose the new state is reset to $|00\ldots0\rangle$ every time.

Since the direct erasure of information is not allowed, it is indeed performed by applying the Hadamard operation on every erased bit and testing whether the result is zero. The testing is *not* performed by a measurement, but the internal error counter is increased in appropriate case. The computation continues regardless until the end. At the end of the computation we measure whether the number of errors is zero. If it is zero, then we measure the result and finish. Otherwise the computation was useless and must be repeated.

To be able to repeat the computation we need to reset the memory, which is done by uncomputation. The final trick is multiplying the current amplitude by -1 in the case that we have not arrived to the starting configuration, it is explained below.

Note 7.3 The simulation could be made in much simpler way by observing the QN after every coin flip. It would collapse the QN into computational state every time, but the acceptance probabilities would be preserved. However this method does not fit into the computational model of QN's, where the measurements are more restricted: the computation can continue for at most one measurement outcome while it immediately

stops for the others. The simple method just stated needs to measure the QN and continue in both cases.

Lemma 7.3 Let P be a PBP for input size n with constant length of computation t, with space complexity s and with the only random choice type $\{\frac{1}{2}, \frac{1}{2}\}$. Then P can be interpreted by a QBP P' in linear space and time. Let us also assume that for every output result $\pi \in \Pi$ there is at most one output vertex labelled by π. The ***interpretation*** differs from the simulation in the acceptance probabilities: if the acceptance probability of P yielding $\pi \in \Pi$ on input $x \in \Sigma^n$ is denoted by $p_\pi^P(x)$, the acceptance probabilities of P' will be

$$p_\pi^{P'}(x) = 2^{-st} \left(p_\pi^P(x)\right)^2.$$

Target QBP P' has one additional outcome "unkn" denoting that it does not know the result. Its probability is $p_{\text{unkn}}^{P'}(x) = 1 - \sum_{\pi \in \Pi} p_\pi^{P'}(x)$.

Proof. The states of P' are indexed by the sequence [current state q_1, new state q_2, random bit b, number of errors e, interpretation step]. The starting vertex is $q_0' = [q_0, 0, 0, 0, \text{i.}]$. We suppose the computation of P comprises only of fair coin flips. If, at some instant, P moves directly to another state, it is regarded as if a coin flip was performed and the result was ignored. Now we are sure that every computational path comprises of the same number t of coin flips. A computational step of P is interpreted (in reversible way) by P' in the following way:

1. apply the Hadamard operation H on b,
2. set $q_2 = q_2 \oplus q$, where q is the destination state corresponding to q_1 assuming the result of coin flip was b,
3. exchange q_1 and q_2,
4. perform $H^{\otimes s}$ on q_2 and H on b,
5. increment e iff $q_2 \neq 0^s$ or $b \neq 0$.

At the end of computation, we measure e. If it is nonzero, we return "unkn". Otherwise we know that q_2 and b have been properly erased in every loop, hence the transitions of b have always had positive amplitudes and q_2 have always been set to q indeed, hence the computation of P' has simulated the computation of P. We can measure q_1 and return the appropriate result.

The probability of being lucky is $P[e = 0] = 2^{-(s+1)t}$. Let us assume we have been lucky. The amplitude of an output vertex q is $a(q) = c(q)/2^{t/2}$,

where $c(q)$ is the number of computational paths ending at q (because the amplitude of every single transition is $1/\sqrt{2}$). For every outcome $\pi \in \Pi$ there is a unique output vertex q_π labelled by π. We know that $p_\pi = c(q_\pi)/2^t$. Hence $a(q_\pi) = 2^{t/2} p_\pi$. We conclude that the probability of obtaining q_π is $2^{-(s+1)t} \cdot 2^t p_\pi^2 = 2^{-st} p_\pi^2$.

The space complexity is increased approximately three times (q_2 takes the same space as q_1 and $e \leq \#Q$), and the time complexity is increased five times. □

Note 7.4 We see that the amplitudes of P' somehow mimic the probabilities of P. This is the reason why the final accept probabilities are squared. This impediment can not be avoided by this method.

Let us analyse the accepting probabilities a, b for $\#\Pi = 2$. We know that $a + b = 1$, let $0 \leq a \leq 1/2 \leq b \leq 1$. The target probabilities after rescaling become $a' = a^2/(a^2+b^2)$, $b' = b^2/(a^2+b^2)$.

A simple analysis shows that $a' \leq a$, $b' \geq b$ and the equivalence happens for $a \in \{0, 1/2\}$, i.e. the squared probabilities are more bounded from $1/2$ than the original probabilities. On the other hand if the original probabilities are bounded from 0 and 1, i.e. $a \geq \varepsilon$, $b \leq 1 - \varepsilon$ then the target probabilities are also bounded from 0 and 1, though the ε would also need to be squared.

When we look at the Table 2.1 with most common acceptance modes on page 12, we see that these properties are sufficient to state that the target QN with squared probabilities can accept the same language in the same mode as the source PBP.

Note 7.5 The requirement, that the source PBP has at most one output vertex assigned to every result, arises from the effort to convey the acceptance probabilities by a simple equation. For example if the source PBP yields π in two distinct output vertices p, q, then

$$p_\pi^{P'}(x) = c_1 \cdot \left(p_p^P(x)^2 + p_q^P(x)^2\right) \neq c_1 \cdot \left(p_p^P(x) + p_q^P(x)\right)^2 = c_1 \cdot \left(p_\pi^P(x)\right)^2,$$

thus it would not suffice to use the acceptance probabilities of the source PBP.

Note 7.6 The fact that we require the source PBP P having one random choice type $\{\frac{1}{2}, \frac{1}{2}\}$, does not restrict it very much, since every other random choice type can be simulated by the fair coin flip at little cost. The constant length of computation constraint also does not matter.

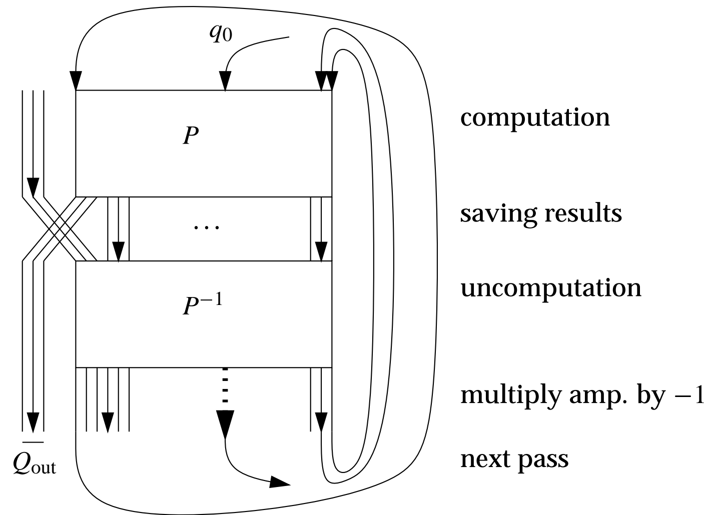


Figure 7.6: Gaining acceptance probabilities of a QBP

Lemma 7.4 Let P be a QBP for input size n with constant length of computation t and with space complexity s. Let $p_\pi^P(x)$ be the probability that P yields $\pi \in \Pi$ on input $x \in \Sigma^n$. Let $\mathrm{unkn} \in \Pi$ (obtaining "unkn" means that the task has not yet been solved) and let us assume that $p_{\mathrm{unkn}}^P(x) \leq 1 - \varepsilon$ and $\varepsilon > 0$ for every x.

Then there exists a layered QN N with space complexity s and expected time complexity $O(t/\varepsilon)$ such that $p_\pi^{P'}(x) = p_\pi^P(x)/(1 - p_{\mathrm{unkn}}^P(x))$ for every $\pi \in \Pi - \{\mathrm{unkn}\}$ and $p_{\mathrm{unkn}}^{P'}(x) = 0$.

Proof. The target network N consists of five layers: computing P, storing results separately, uncomputing P, adjusting the quantum amplitude, and the measurement plus the identity transformation provided by the back-edges. Everything is outlined in Figure 7.6.

Since the measurement is performed in the last layer, we add $t \cdot \#Q_{out}$ vertices to the left side of P to carry the amplitudes. After performing the computation of P we exchange the output vertices with zero vertices having zero amplitudes — it behaves as if a measurement yielding the result "unkn" has been performed. Then we uncompute P, correct the amplitude of q_0 (notice that $q_0 \notin Q_{in}$, however it is allowed) and perform the deferred measurement. It is obvious that if we are lucky and the result is found in first iteration then the probabilities are scaled exactly as stated.

Let us investigate what happens when we obtain "unkn" instead, i.e. the measurement collapses N to the state as if the measurement with nega-

tive outcome has been performed at the time when the results were saved. We immediately obtain an important remark: if we do not perform the amplitude correction in the fourth layer, the sequence $P^{-1}P$ would behave exactly as the identity operator, hence the computation of N would never finish yielding "unkn" at the end of every iteration. The correction can be regarded as a small kick that perturbs the state enough to leave an unwanted vector subspace.

However we have to compute exactly what happens. N begins in state $|q_0\rangle$. After P if launched, the target state $|\psi\rangle = U_x|q_0\rangle$ can be expressed as $|\psi\rangle = \sum_{\pi\in\Pi}|\psi_\pi\rangle$, where $|\psi_\pi\rangle$ is a projection of $|\psi\rangle$ onto the subspaces spanned by $\{|q\rangle | q \in Q_{\text{out}} \; \& \; d(q) = \pi\}$. We know that $\| \, |\psi_\pi\rangle \, \|^2 = p_\pi^P(x)$.

Let us suppose that N has collapsed to $|\psi_{\text{unkn}}\rangle$, i.e. we have not obtained the result in first iteration. Let $\Pi' = \Pi - \{\text{unkn}\}$. After P is uncomputed, the target state becomes

$$\begin{aligned} |q_1\rangle = U_x^{-1}|\psi_{\text{unkn}}\rangle &= U_x^{-1}\left(|\psi\rangle - \textstyle\sum_{\pi\in\Pi'}|\psi_\pi\rangle\right) \\ &= |q_0\rangle - \textstyle\sum_{\pi\in\Pi'} U_x^{-1}|\psi_\pi\rangle, \end{aligned}$$

ignoring the fact that another block of computation has been performed, i.e. a component of the internal state is different. Let us choose $|\xi_\pi\rangle = U_x^{-1}|\psi_\pi\rangle - p_\pi^P(x)|q_0\rangle$ deliberately such that $\langle q_0|\xi_\pi\rangle = 0$. We express $|q_1\rangle$ using these new variables and after the amplitude correction on states orthogonal to $|q_0\rangle$ is performed, the computational state becomes $|q_2\rangle$.

$$\begin{aligned} |q_1\rangle &= \left(1 - \textstyle\sum_{\pi\in\Pi'} p_\pi(x)\right)|q_0\rangle - \textstyle\sum_{\pi\in\Pi'}|\xi_\pi\rangle, \\ |q_2\rangle &= p_{\text{unkn}}(x)|q_0\rangle + \textstyle\sum_{\pi\in\Pi'}|\xi_\pi\rangle. \end{aligned}$$

We enter second iteration: after P is launched again, the state becomes

$$\begin{aligned} |q_3\rangle &= p_{\text{unkn}}(x)U_x|q_0\rangle + \textstyle\sum_{\pi\in\Pi'} U_x|\xi_\pi\rangle \\ &= p_{\text{unkn}}(x)\left(\textstyle\sum_{\pi\in\Pi}|\psi_\pi\rangle\right) + \textstyle\sum_{\pi\in\Pi'}\left(|\psi_\pi\rangle - p_\pi(x)\left(\textstyle\sum_{\rho\in\Pi}|\psi_\rho\rangle\right)\right) \\ &= 2p_{\text{unkn}}(x)\textstyle\sum_{\pi\in\Pi'}|\psi_\pi\rangle + \left(1 - 2\textstyle\sum_{\pi\in\Pi'} p_\pi(x)\right)|\psi_{\text{unkn}}\rangle, \end{aligned}$$

hence the amplitude of $|\psi_{\text{unkn}}\rangle$ is $c = (1 - 2\sum_{\pi\in\Pi'} p_\pi(x))$ times larger than it was at first iteration. Therefore at the third, fourth, ..., i-th iteration all amplitudes will be c^{i-1} times larger, i.e. the state after the program P is launched for the i-th time ($i \geq 2$) will be

$$2p_{\text{unkn}}(x) \cdot c^{i-2} \sum_{\pi\in\Pi'} |\psi_\pi\rangle + c^{i-1}|\psi_{\text{unkn}}\rangle.$$

Hence the probability that N ever accepts π is calculated as (adding the probability from the first iteration)

$$
\begin{aligned}
p_\pi^{P'}(x) &= p_\pi^P(x) + 4\left(p_{\text{unkn}}(x)\right)^2 \textstyle\sum_{i=2}^\infty \left(c^2\right)^{i-2} \| \, |\psi_\pi\rangle \, \|^2, \\
\textstyle\sum_{i=0}^\infty (c^2)^i &= \textstyle\sum_{i=0}^\infty \left(1 - 4\sum_{\pi\in\Pi'} p_\pi(x) + 4\left(\sum_{\pi\in\Pi'} p_\pi(x)\right)^2\right) \\
&= \frac{1}{4\sum_{\pi\in\Pi'} p_\pi(x) - 4\left(\sum_{\pi\in\Pi'} p_\pi(x)\right)^2} = \frac{1}{4\sum_{\pi\in\Pi'} p_\pi(x)\left(1-\sum_{\pi\in\Pi'} p_\pi(x)\right)} \\
p_\pi^{P'}(x) &= p_\pi(x) + p_\pi(x)\frac{4(p_{\text{unkn}}(x))^2}{4\sum_{\pi\in\Pi'} p_\pi(x) p_{\text{unkn}}(x)} \\
&= p_\pi(x)\left(1 + \frac{p_{\text{unkn}}(x)}{1-p_{\text{unkn}}(x)}\right) = \frac{p_\pi(x)}{1-p_{\text{unkn}}(x)},
\end{aligned}
$$

which is the desired result. The expected time can be computed from the same equation using creating functions:

$$
\begin{aligned}
\text{ExpTime}(N) &= \textstyle\sum_{\pi\in\Pi'} p_\pi^P(x)\left[1 + 4\left(p_{\text{unkn}}(x)\right)^2 \cdot \sum_{i=0}^\infty (i+2)(c^2)^i\right], \\
f(x) &= \textstyle\sum_{i=0}^\infty \left(c^2 x\right)^i = \frac{1}{1-c^2x}, \\
f'(x) &= \textstyle\sum_{i=1}^\infty i\left(c^2x\right)^{i-1} c^2 = c^2 \sum_{i=0}^\infty (i+1)\left(c^2x\right)^i \\
&= \frac{-(-c^2)}{(1-c^2x)^2} = \frac{c^2}{(1-c^2x)^2}, \\
\text{ExpTime}(N) &= \left(1 - p_{\text{unkn}}(x)\right)\left[1 + 4\left(p_{\text{unkn}}(x)\right)^2 \cdot \left(f(1) + \frac{f'(1)}{c^2}\right)\right].
\end{aligned}
$$

We know that $\sum_{\pi\in\Pi'} p_\pi^P(x) \geq \varepsilon$, $p_{\text{unkn}}^P(x) \leq 1-\varepsilon$, and $c \leq 1-2\varepsilon$. Let us assume the inequalities are equalities, as this would be the worst case. We substitute these equations:

$$
\begin{aligned}
\text{ExpTime}(N) &= \varepsilon\left[1 + 4(1-\varepsilon)^2\left(\frac{1}{1-c^2} + \frac{c^2}{c^2(1-c^2)^2}\right)\right] \\
&= \varepsilon + 4\varepsilon(1-\varepsilon)^2\left(\frac{1-c^2+1}{(1-c^2)^2}\right), \\
c^2 &= 1 - 4\varepsilon + 4\varepsilon^2, \\
\text{ExpTime}(N) &= \varepsilon + 4\varepsilon(1-\varepsilon)^2 \frac{1+4\varepsilon-4\varepsilon^2}{(4\varepsilon(1-\varepsilon))^2} \\
&= \varepsilon + \frac{1+4\varepsilon-4\varepsilon^2}{4\varepsilon} = \frac{4\varepsilon^2+1+4\varepsilon-4\varepsilon^2}{4\varepsilon} \\
&= \frac{1+4\varepsilon}{4\varepsilon} \leq \frac{1+4}{4\varepsilon} = \frac{5}{4\varepsilon}, \\
\text{ExpTime}(N) &= \mathrm{O}(1/\varepsilon). \qquad \text{(in iterations)}
\end{aligned}
$$

We have constructed a layered QN N working in space s and expected time $\mathrm{O}(t/\varepsilon)$ (however not halting absolutely), that never yields "unkn" and the probabilities of other outcomes correspond to the probabilities of source QBP P appropriately scaled. □

Theorem 7.5 Every PBP P with time complexity t can be interpreted by a QN N with the same space complexity and with expected time complexity

$O(t \cdot 2^{st})$. N does not halt absolutely. If the source acceptance probabilities are $p^P_\pi(x)$, the target acceptance probabilities are $p^N_\pi(x) \equiv (p^P_\pi(x))^2$.

The theorem holds also for sequences of PBP's.

Proof. Let us just sketch the proof. We first need to modify P to have just one random choice type (fair coin flip), which can be done with a constant time overhead. Then we justify all computational paths of P to have equal length, it can be done by adding a counter to the internal state, which does not increase the space complexity very much. Then we merge all output vertices labelled by the same output letter π. We apply Lemma 7.3 and obtain a QBP P' with acceptance probabilities squared and decreased 2st times. Since the output alphabet Π is finite, one of the acceptance probabilities of P is big enough:

$$(\exists \pi \in \Pi) p^P_\pi(x) > 1/\#\Pi,$$

hence the sum of the target probabilities is not less than $1/(2^{st}\#\Pi)$. We apply Lemma 7.4 and obtain a target QN N with time complexity $O(t \cdot 2^{st})$.

This theorem obviously holds for nonuniform sequences. To prove it also for uniform sequences, let us outline a target construction machine C'. It will comprise of three machines C'_1 (justifying P), C'_2 (constructing P') and C'_3 (constructing N).

The justification of P: replacing the complex random choice types by the chain of simple ones can be done easily, since we need not take care of the interference pattern. Stretching the computational paths to unique length can be done by adding a counter to the internal state.

The intermediate QBP P' is very regular: its internal states have simple structure, description of almost all computational steps can be computed using simple arithmetics, the encoding of the computational step performing the desired operation is obtained by asking for advice.

The target QN N is also very regular: it comprises of two copies of P' (one of them reversed) and a few additional vertices and edges. Every bit of the encoding can be computed using simple arithmetics and asking for advice.

We conclude (using the same arguments that have been used in previous theorems) that the construction machine C' has space complexity $O(s(n))$ and time complexity $O(p(s(n)))$. □

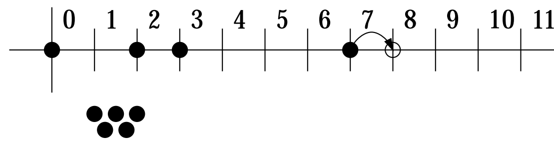


Figure 7.7: Scheme of the Pebble game

7.3 The Pebble game method

Both methods introduced increase the time complexity exponentially. Let us search a method achieving faster reversible programs.

We ignore the simplest method that remembers the complete history of computation. This method is apparently reversible since it never forgets anything, it rather allocates a new storage space for every computational step. This method increases the space complexity too much.

However the notion of remembering a part of the history of computation is very useful. To decrease the space complexity we should not remember every instant of computation, but only a few discrete places. We exploit the fact the the other instants can be always recomputed from the nearest place when needed. This increases the time complexity, but if we implement this method well, both time overhead and space overhead will be polynomial.

Let us reformulate the method as a game.

Game 7.1 *Pebble game*: We are given $n+1$ pebbles. One of them lies permanently on the number 0, the other n pebbles are on a pile. At every instant we can either put a pebble on a natural number i or pick a pebble lying on i up. However both operations can be done only if there is another pebble already lying on $i-1$. The question is: What is the largest number d_n, which can be covered by a pebble? How many operations t_n need to be performed to reach d_n?

Claim 7.6 $d_n = 2^n - 1$, $t_n = (3^n - 1)/2$.

Proof. Let us prove that $d_n \geq 2^n - 1$, i.e. show a particular procedure reaching d_n. We construct recursively a procedure $P(n)$. It gets n pebbles for its disposal and has to reach the most distant number d_n. It works in the following way:

1. if $n = 0$, it can do nothing and stops the job, reaching position $d_0 = 0$,

2. call $P(n-1)$ to use all but one pebble and shift the current position d_{n-1} numbers to the right,

3. place the last pebble on the furthest reachable number $A = d_{n-1} + 1$,

4. call the inverse of $P(n-1)$ to collect the $n-1$ pebbles back from the plane into the pile,

5. call again $P(n-1)$ to use all spare pebbles and thus shift the current position d_{n-1} numbers to the right again, now starting from A.

We see that $P(n)$ reaches $d_n = 2d_{n-1} + 1$ using $t_n = 3t_{n-1} + 1$ operations. The equations can be proved by mathematical induction. It holds that $d_0 = t_0 = 0$. The second step of induction:

$$\begin{aligned} d_n &= 2d_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1, \\ t_n &= 3t_{n-1} + 1 = 3(3^{n-1} - 1)/2 + 1 = (3^n - 3 + 2)/2 = (3^n - 1)/2. \end{aligned}$$

Let us prove $d_n \leq 2^n - 1$, i.e. the method can not be improved. We can do this also by induction. The first step $d_0 \leq 0$ is obvious. The second step: let us look at the first instant when all pebbles are put on the line and let A be the largest reached number. Even if we gather all pebbles except the last one lying on A, we can not reach further number than $A + d_{n-1}$. If we know that $d_{n-1} \leq 2^{n-1} - 1$, if follows that $A \leq 2^{n-1}$ and thus $d_n \leq 2^n - 1$. □

Let us apply the results for reversible computation. A pebble can be regarded as a storage space for one instant of computation: If it lies on number i, it means we remember the state of computation after the i-th step. If it is on the pile it means the storage space is empty. Putting a pebble on number i when $i-1$ is covered corresponds to computing the i-th step of computation by running the source program P, because the state of computation after the $(i-1)$-th step is loaded in memory. The irreversibility of P does not matter, since we store the results into a new clear storage space. Further, picking a pebble up from number i when $i-1$ is covered corresponds to uncomputing the results of the i-th step of computation and hence clearing the storage space.

Theorem 7.7 Let P be a layered BP with space complexity s and time complexity t. Let w denote the width of the widest layer of P. Then there exists a layered RBP P' equivalent with P with space complexity $O(\log w \cdot \log t) = O(s^2)$ and time complexity $O\left(t^{\log_2 3}\right) = O\left(t^{1.585}\right)$.

Proof. Let $p = \lceil \log_2 t \rceil$ be the number of pebbles and let $\mathbf{Z}_w = \{0, 1, \ldots, w-1\}$ be the storage space big enough to store the computational state in any layer. The memory of P' will comprise of p copies of $\mathbf{Z}_w$, i.e. $Q' \equiv \mathbf{Z}_w^p$. To perform the computation of P in reversible way, we follow the Pebble game. Notice that p has been chosen such that $t = \mathrm{O}(2^p)$ is a reachable computational step.

The space complexity of P' is $p \log_2 w = \mathrm{O}(\log w \cdot \log t)$, both $t, w \leq \#Q = \mathrm{O}(2^s)$ hence the space complexity of P' is $\mathrm{O}\left(s^2\right)$. The time complexity of P' is $\mathrm{O}(3^p)$. We know that $3^{\log_2 t} = 3^{\log_3 t / \log_3 2} = \left(3^{\log_3 t}\right)^{\log_2 3} = t^{\log_2 3}$.

It remains to be shown that a computational step of BP P can be simulated by RBP P' under an assumption that the results are stored into cleared memory. Let $[q_1, q_2, \ldots, q_p]$, $q_i \in \mathbf{Z}_w$ be the current state of P'. To perform the l-th computational step on state stored at index a and store it to index b, the following operation is performed:

$$q_b := q_b \oplus P^{(l)}(q_a),$$

where $P^{(l)}$ is the extended operation performed by P in l-th layer. $P^{(l)}$ is extended (e.g. by zeroes) to be defined for all $q_a \in \mathbf{Z}_w$, since the original BP P can have narrower layers. It is clear that such operation is reversible and moreover that it is its own inversion. Hence it can serve also for the uncomputation. □

This method obviously works for nonuniform sequences without need of change.

Theorem 7.8 Every uniform sequence of layered BP's P constructed by a machine C can be converted into a uniform sequence of layered RBP's P' with the overhead and constraints mentioned in Theorem 7.7.

Let s be the space complexity of P. The construction machine C' of P' has space complexity linear in s and time complexity polynomial in s.

Proof. Let us use similar encoding of the source program P as in the proof of Theorem 7.2 on page 85. We also require the source encoding contains the width of the layer and the number of layers in the header and the source vertices are indexed consecutively along the layers (see Note 6.2 on page 66, the same requirements have simplified the proof of Theorem 6.5 on page 68).

Under these assumptions the conversion of P to P' involves just simple arithmetics and it comprises of a few inquiries for an advice:

- target vertices Q' will be indexed by the pair [layer number l', index of the vertex in the layer v'], target families will be indexed in

similar way by the pair [layer l', index of its leftmost vertex v'] independently on the source family indexing (we must be cautious since there will be many holes in the family indexing),

- when working with target vertex $q' = [l', v']$, our first task is to translate the target layer number l' to triple l, a, b, where l is the number of source layer and $a \to b$ are the indexes of the internal storage space, it can be done using simple arithmetics by expanding l' in basis 3 and following the behaviour of the Pebble game algorithm,
- when working with target family $f' = [l', v']$, first task after expanding l' to l is to check whether such family exists (done by asking the advice for the family of $[l, v']$ and checking whether v' is the leftmost vertex of its family), all other steps also imply from the arithmetics and asking the advice,
- after the behaviour $P^{(l)}$ of P in l-th layer is obtained from the advice, it is not difficult to transform it into the operation $q_b := q_b \oplus P^{(l)}(q_a)$ using simple arithmetics: we know l', l, a, b, all activity is determined by the vertex q_a (q_b is uninteresting since it is just XOR-ed).

All numbers C' works with must be big enough to carry the index of a vertex, a layer or a family, hence the space complexity of C' is $O(s(n))$ and the time complexity of C' is $O(p(s(n)))$. □

It is tempting to state a similar theorem also for PBP's — replacing fair coin flips by Hadamard operations (avoiding negative amplitudes by ensuring the argument is $|0\rangle$) and achieving reversibility by storing the results separately and then uncomputing the arguments. Unfortunately this does not work so easily due to the interference pattern.

Look at Figure 7.8. The source PBP has only one irreversible step in the third layer with destination set of size 2. Hence we need not convert the target QBP exactly using the Pebble game algorithm, but we can shortcut the construction a little: first two layers are directly replaced by a set of Hadamard operations, the third layer is simulated exactly is described above, and finally the uncomputation of first two layers finishes the computation. Therefore only two copies of the graph need to be involved.

Let us investigate the progress of computation of the target QBP, which is described in Table 7.1. Every column corresponds to the amplitude of a vertex in current layer. We see that separating an amplitude in the third layer perturbs the amplitude distributions in either copies of the graph in such a way that the uncomputation yields neither the desired distribution $[\sqrt{3/4}, \sqrt{1/4}]$ nor its pure squared variant $[c \cdot 3/4, c \cdot 1/4]$, but something

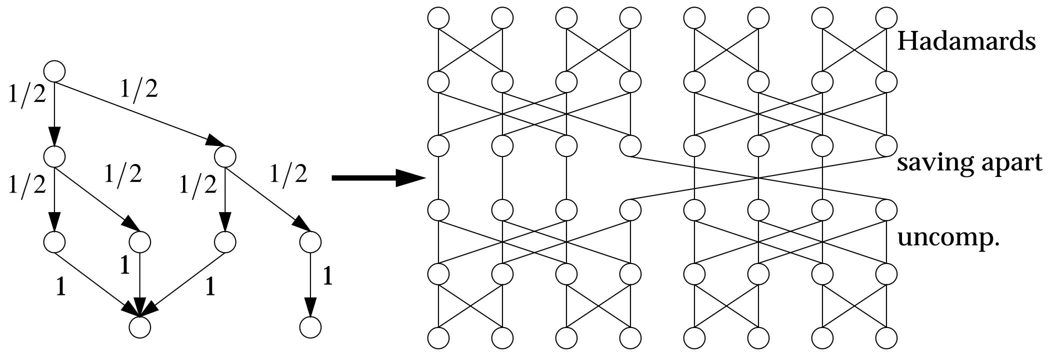


Figure 7.8: A PBP converted using Pebble game

layer	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0
2	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	0
3 — uniform	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
4 — permuted	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	$\frac{1}{2}$
5	$\frac{1}{\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	0	$\frac{1}{2\sqrt{2}}$	0	$\frac{1}{2\sqrt{2}}$	0	$\frac{-1}{2\sqrt{2}}$
6	$\mathbf{\frac{3}{4}}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{-1}{4}$	$\mathbf{\frac{1}{4}}$	$\frac{-1}{4}$	$\frac{-1}{4}$	$\frac{1}{4}$
probabilities	$\mathbf{\frac{9}{16}}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\mathbf{\frac{1}{16}}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$

Table 7.1: Computation of the PBP in Figure 7.8

more chaotic: the squared variant $[3/4, 1/4]$ justified by some 'random' amplitudes of the other vertices. It turns out that this is not casual, but it is the rule. Anyway, this chaotic distribution is useless as an intermediate state of the computation.

Let us investigate in detail what indeed happens to the amplitudes. Let r denote the number of fair coin flips and $x \bullet y$ be the bitwise scalar product.

- the fair coin flips distribute the amplitude of the starting vertex uniformly into $2^{-r/2} \sum_{t \in Q} |0, t\rangle$ in the third layer,
- after the vertices are permuted, the uncomputation of the fair coin flips distributes the amplitude of every vertex $|q, t\rangle$ in the fourth layer into $2^{-r/2} \sum_{i \in Q} (-1)^{i \bullet t} |q, i\rangle$; the sign is always $+1$ for $|q, 0\rangle$, hence $|q, 0\rangle$ gains total amplitude $2^{-r} c(q) = p_q^P(x)$, where $c(q)$ is the number of vertices $|q, t\rangle$ from the fourth layer,

- however due to the vertex permutation in the third layer, the amplitudes in the fourth layer do not form the desired pattern and the uncomputation of fair coin flips does not abate the randomness by interfering destructively the amplitudes of vertices $|q,i\rangle$, $i \neq 0$; their amplitudes can be regarded as an unwanted noise,

- even if we get rid of the unwanted amplitudes, we would have to consider that the resulting probability distribution is squared.

One way of circumventing this impediment is by avoiding the uncomputation of coin flips. If we flip all coins in the beginning of the computation and remember the results, the rest of the computation of the PBP can be performed deterministically — the i-th coin flip is replaced by a conditional jump depending on the i-th stored result.

Theorem 7.9 Let P be a layered PBP with space complexity s, with time complexity t and with the only random choice types $\{\frac{1}{2}, \frac{1}{2}\}$ (fair coin flip) and $\{1\}$ (deterministic step). Let w denote the width of the widest layer of P and r denote the maximal number of random choices on a computational path. Then there exists a layered QBP P' equivalent with P with space complexity $\mathrm{O}(r + \log w \cdot \log t) = \mathrm{O}\left(r + s^2\right)$ and time complexity $\mathrm{O}\left(r + t^{\log_2 3}\right) = \mathrm{O}\left(t^{1.585}\right)$.

Proof. The target QBP P' has memory big enough to remember r results $y = \{y_i\}_{i=1}^r$ of coin flips and $p = \lceil \log_2 t \rceil$ computational states in any layer. The computation of P' starts with applying the Hadamard operation on every qbit y_i, this can be done in r layers (with constant size families).

Henceforth the source PBP P is regarded as a BP P^d that comprises of only deterministic (though irreversible) steps. The i-th fair coin flip is replaced by a deterministic transition to the vertex corresponding to the result stored in y_i. We convert the BP P^d into a RBP $P^{d'}$ using Theorem 7.7 and include it in P'.

Hence P' finishes in state $[y_1, y_2, \ldots, y_r, 0, \ldots, 0, c, 0, \ldots 0]$, $y_i \in \{0,1\}$, c is such that $q_{t,c} = P^{d'}(x,y) \in Q_{\mathrm{out}}$ (uniquely determined by x,y) with amplitude $2^{-r/2}$. If we measure c, the probability $p_\pi^{P'}(x)$ of observing a vertex labelled by $\pi \in \Pi$ is equal to $2^{-r} \cdot \sum_{y \in \{0,1\}^r} p_\pi^{P^{d'}}(x,y) = p_\pi^P(x)$. We conclude that P' simulates (not interprets) P. □

Note 7.7 Since $r = \mathrm{O}(t)$, the target QBP P' has space complexity $\mathrm{O}\left(s^2 + t\right)$ while the naive method (remembering complete history of computation) would lead to an algorithm with space complexity $\mathrm{O}(st)$. It is slightly

better, but since it can happen that $r = \Theta(t)$ and $t = \Theta(2^s)$, the space complexity of P' could reach $O\left(s^2 + 2^s\right)$.

This method obviously works for nonuniform sequences without need of change.

Theorem 7.10 Every uniform sequence of layered PBP's P with simple structure of random choices constructed by a machine C can be converted into a uniform sequence of layered QBP's P' with the overhead and constraints mentioned in Theorem 7.9. We say that a PBP P_n has ***simple structure of random choices*** iff it uses only fair coin flips and the rank of the random choice can be computed from the vertex identifier in little space and time.

Let s be the space complexity of P. The construction machine C' of P' has space complexity linear in s and time complexity polynomial in s.

Proof. The first phase of P'_n (applying Hadamard operations on stored random qbits) is very regular and can be generated by C' without problems. The second phase (reversible simulation of P^d_n) is constructed in the same way as in the proof of Theorem 7.8. We only need to obtain the deterministic BP P^d_n from the source PBP P_n, which is also straightforward, since random choices of P_n have a simple structure. □

7.4 Tradeoffs between the methods

We have developed more methods achieving reversibility, two of them preserve space and increase time too much, another one almost preserves time and increases space a little. It is also possible to combine these methods and obtain a variety of algorithms with scaled space and time complexities. Some tradeoffs of this type have been already independently explored in [Ben89].

Let us imagine that we convert a source BP P_1 to another BP P_2 using the Pebble game method. If we are given enough storage space, P_2 could be made reversible directly and the job is finished. Otherwise only blocks of the computation comprising of $l < t$ consecutive layers can be made reversible. The i-th block starts in the state $[q_i, 0, 0, \dots]$ and it leaves P_2 in the state $[q_i, q_{i+1}, 0, \dots]$. Since the reversible computation of every block needs to get all memory items cleared, we must erase q_i and replace it by q_{i+1} before the computation of the $(i+1)$-th block is launched. Hence P_2 comprises of $\lfloor t/l \rfloor$ irreversible layers interpolated by a reversible computation.

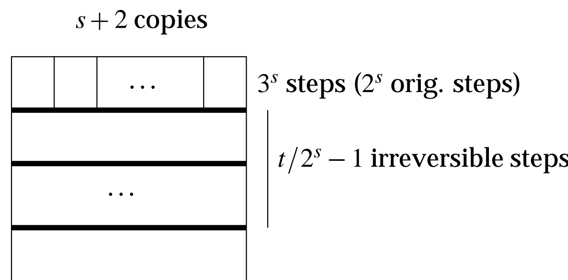


Figure 7.9: Tradeoff between the Pebble game and back-tracking

If we convert P_2 to a RBP P_3 using the back-tracking method, we obtain a RBP working in the same space.

Theorem 7.11 Let P be a layered BP with time complexity t. Let w denote the width of the widest layer of P. Then for every $s \in \{0,1,2,\ldots,\lceil \log_2 t \rceil\}$ there exists a layered RBP P_s equivalent with P with space complexity $O((s+2)\cdot \log w)$ and time complexity $O\left((3/2)^s \cdot t \cdot w^{t/2^s-1}\right)$.

The theorem holds also for sequences of BP's. The construction machine of the target uniform sequence has little space and time complexities.

Proof. Let us fix the value of the parameter s. Since the first memory position is reserved for the input value, we have $s+1$ free memory positions. Using Claim 7.6, we see that the Pebble game can reach position $2^{s+1}-1$ using $(3^{s+1}-1)/2$ movements. However we want to empty all memory items besides the input and output value, hence we stop playing the Pebble game at the place A in the middle of the first recursive call, thus only position 2^s is reached and it takes 3^s movements.

The computation of the intermediate BP P^g consists of $\lceil t/2^s \rceil$ blocks. The computation of every block comprises of 3^s layers obtained by the reversible simulation of the corresponding block of P. After each block except the last one the contents of the input memory item is replaced by the output memory item (this operation is not reversible). See Figure 7.9.

The target RBP P_s is obtained by back-tracking P^g. There are $O(3^s \cdot t/2^s)$ layers and $t/2^s-1$ of them are irreversible. The back-tracking just passes through at a reversible layer, but it explores all input edges at an irreversible layer. Since one memory item is erased in such layer, every vertex has w incoming edges there. We conclude that $w^{t/2^s-1}$ branches of length $(3/2)^s \cdot t$ are explored.

Since we have already proved that the construction machines for both back-tracking and the Pebble game method have little complexities, so has the construction machine C' of the combined method. The machine C' is just a composition of two machines: C'_1 constructing P^g and C'_2 constructing P'. C'_2 has already been discussed, C'_1 is a simple variant of the original Pebble game construction. □

It is also possible to form a tradeoff for the interpretation of PBP's. It comprises of converting large blocks of the source program by the Pebble game method, concatenating the consecutive blocks into a global QBP, getting rid of unwanted intermediate amplitudes by counting a number of errors, and iterating the computation until we are lucky and observe that no errors have occurred.

Let us describe in detail how the conversion is performed. Given the space complexity of the target QBP P', we choose the block size l as large as possible and also allocate a space in P' for an error counter. We divide the computation of the source PBP P into blocks $P^{(i)}$ of size l and convert the i-th block using the Pebble game method into a QBP $P^{g,(i)}$. The QBP $P^{g,(i)}$ starts in the state $[0^l, q_i, 0, 0, \ldots]$ and ends in the state $[y_1, \ldots, y_l, q_i, q_{i+1}, 0, \ldots]$. We uncompute l fair coin flips (by applying Hadamard operations again) and try to erase q_i also by applying Hadamard operations. If we have not arrived to a state in the form $[0^l, 0, x, 0, \ldots]$, we increment the error counter. Otherwise we permute the components of the internal state into $[0^l, q_{i+1}, 0, 0, \ldots]$. The global QBP P^g comprises of a concatenation of the individual $P^{g,(i)}$'s. It measures the error counter at the end and yields "unkn" if it is nonzero. Otherwise it measures also the output vertices and finishes. The output probabilities will be squared. The target QBP P' will iterate P^g while "unkn" has been observed.

Theorem 7.12 Let P be a layered PBP with time complexity t and with the only random choice types $\{\frac{1}{2}, \frac{1}{2}\}$ and $\{1\}$. Let w denote the width of the widest layer of P. Let us also assume that for every result $\pi \in \Pi$ there exists at most one output vertex labelled by π. Then for every value of the parameter $s \in \{0, 1, 2, \ldots, \lceil \log_2 t \rceil\}$ there exists a layered QN P_s interpreting P with space complexity $O(2^s + (s+2) \cdot \log w)$ and expected time complexity $O\left((3/2)^s \cdot t \cdot w^{t/2^{s-1}}\right)$. It, however, does not halt absolutely.

The theorem holds also for sequences of PBP's. The construction machine of the target uniform sequence has little space and time complexities.

Proof. Let us fix the value of the parameter s. The target QN P_s will work as described above and in the proof of Theorem 7.11. To be able to simulate in reversible way the 2^s consecutive layers of the source PBP P, it

remembers the computational state in $(s+2)$ independent layers, 2^s results of the coin flips, and an error counter. Since the number of errors is not bigger than $t/2^s$ and the target layer number is less than $(3/2)^s \cdot t$, the space complexity of P is $\mathrm{O}(2^s + (s+2) \cdot \log w)$.

The time complexity of the intermediate QBP P^g is $(3/2)^s \cdot t$. The probability that the number of errors will not be increased in a layer erasing the memory items is at least $1/w^2$, because every bit of configuration q_i can evolve into 0 or 1 and there are $\log_2 w$ such bits, and the probability that the fair coin flips are properly cleared is $\sum_{q \in Q^{(i+1)}} (p_q^{P^{(i)}}(x))^2 > 1/w$ (it follows from $\sum_{q \in Q^{(i+1)}} p_q^{P^{(i)}}(x) = 1$ and $\#Q^{(i+1)} \leq w$ by applying the arithmetical-quadratical inequality). Since there are $t/2^s$ erasing layers, we conclude that the probability of observing no errors is $(1/w^2)^{t/2^s} = 1/w^{t/2^{s-1}}$. The expected time complexity of the target QBP P' is the product of the time complexity of P^g and the inverse of the probability of being lucky.

Let us show that P' interprets P. Assuming no errors have been observed, we know that the amplitudes of the intermediate vertices at the erasing layers are proportional to their probabilities in the source program P. We know that every output letter is assigned to less than one vertex, hence the probabilities are $p_\pi^{P'}(x) = (p_\pi^P(x))^2$.

The construction machine C' is apparently composed from the construction machines C'_1 (playing the Pebble game), C'_2 (concatenating the individual QBP's, erasing the allocated space and counting the number of errors), and C'_3 (iterating the computation until we are lucky). □

Chapter 8

Space bounded Quantum Computation

We state two interesting related results in this chapter. The first result described in [Bar89] says that width-5 oblivious RBP's recognise NC_1 languages in polynomial time. The second one described in [AMP02] claims the same about width-2 oblivious QBP's. Hence a quantum memory comprising just of one qbit is big enough to perform a useful computation.

8.1 Recognising NC_1 languages by 5-PBP's

Definition 8.1 Let $L \subseteq \{0,1\}^*$ be a language. We say that $L \in \mathrm{NC}_k$, if there exists a sequence of circuits $C = \{C_n\}_{n=1}^{\infty}$ such that C_n recognises $L_n = L \cap \{0,1\}^n$ (it yields 1 iff $x \in L_n$). C_n is a Boolean circuit with n inputs x_1, x_2, $\dots$, x_n comprising of AND, OR and NOT gates of fan-in 2 and it has depth $\mathrm{O}\left(\log^k n\right)$. If such sequence is uniform (the encodings of C_n's can be constructed by a TM), we say that $L \in \text{uniform } \mathrm{NC}_k$.

Definition 8.2 We say that P is a ***width-w permutation Branching Program (w-PBP)***, if it is an oblivious (i.e. layered with one decision variable in every layer) reversible BP having exactly w vertices in every layer.

It is straightforward that a w-PBP performs a permutation on vertices in every layer. Hence a w-PBP can be regarded as a machine composing permutations.

D. A. Barrington has proved in [Bar89] that for every $L \in \mathrm{NC}_1$ there exists a sequence P of 5-PBP's of polynomial length recognising L and vice versa. If L is uniform then P can also be taken uniform. Let us outline the method.

Definition 8.3 We say that a 5-PBP P ***five-cycle recognises*** a language $L \subseteq \{0,1\}^n$ if there exists a five-cycle σ (called the ***output***) in the permutation group S_5 such that $P(x) = \sigma$ if $x \in L$ and $P(x) = 1$ if $x \notin L$ (1 is the identity permutation).

Theorem 8.1 Let L be recognised by a depth d fan-in 2 Boolean circuit. Then L is five-cycle recognised by a 5-PBP P of length at most 4^d.

Lemma 8.2 If P five-cycle recognises L with output σ and τ is any five-cycle, then there exists a 5-PBP P' recognising L with output τ. It has the same length as P.

Proof. Since σ and τ are both five-cycles, there exists a permutation θ such that $\tau = \theta\sigma\theta^{-1}$. To get P' we simply take P and for every layer i, we replace either permutations α_i and β_i (performed respectively when the value of the decision variable is 0 and 1) by $\theta\alpha_i\theta^{-1}$ and $\theta\beta_i\theta^{-1}$ in the i-th layer. □

Lemma 8.3 If L is five-cycle recognised in length l, so is its complement $L' = \{0,1\}^n - L$.

Proof. Let P be a 5-PBP recognising L with output σ. Let us take its last layer and replace either permutations α_i and β_i by $\alpha_i\sigma^{-1}$ and $\beta_i\sigma^{-1}$. We denote the target 5-PBP by P'. Then $P'(x) = 1$ if $x \in L$ and $P'(x) = \sigma^{-1}$ if $x \notin L$, hence P' five-cycle recognises L'. □

Lemma 8.4 There exist two five-cycles $\varphi_1, \varphi_2 \in S_5$ whose commutator is a five-cycle. (The commutator of a and b is $aba^{-1}b^{-1}$.)

Proof. Take $\varphi_1 = (12345)$, $\varphi_2 = (13542)$. Then $\varphi_1\varphi_2\varphi_1^{-1}\varphi_2^{-1} = (13254)$. □

Proof of Theorem 8.1. By induction on the depth d: If $d = 0$, the circuit is just an input gate and L can easily be recognised by an one instruction 5-PBP. Assume w.l.o.g. that $L = L_1 \cap L_2$, where L_1, L_2 have circuits of depth $d-1$ and thus are recognised by 5-PBP's P_1, P_2 of length at most 4^{d-1} (use Lemma 8.3 for the implementing NOT and OR gates). Let P_1, P_2 have outputs φ_1, φ_2 from Lemma 8.4 and P_1', P_2' have outputs $\varphi_1^{-1}, \varphi_2^{-1}$ (it can be prescribed by Lemma 8.2).

Let P be the concatenation $P_1P_2P_1'P_2'$. P yields 1 if $x \notin L_1 \cap L_2 = L$, but it yields the commutator of φ_1, φ_2 if $x \in L$. The commutator is a five-cycle, hence P five-cycle recognises L. Moreover P has length at most $4 \cdot 4^{d-1} = 4^d$. Given a circuit and a desired output, this proof gives a deterministic method of constructing the 5-PBP. □

Theorem 8.5 Let $L \subseteq \{0,1\}^n$ be recognised by a w-PBP P of length l. Then L is recognised by a fan-in 2 circuit C of depth $\mathrm{O}(\log l)$, where the constant depends on w.

Proof. A permutation of w items can be represented by $\mathrm{O}(w^2)$ Boolean variables telling whether $f(i) = j$ for every i, j. The composition of two permutations can be performed by a constant depth circuit. The target circuit C will comprise of a constant depth section composing a permutation yielded by each instruction of P, a binary tree of composition circuits, and a constant depth section at the top determining the acceptance given the permutation yielded by P. □

Corollary 8.6 The w-PBP's (for $w \geq 5$) of polynomial length recognise exactly NC_1 languages. The equivalence holds both for nonuniform and uniform sequences.

8.2 NC_1 is contained in 2-EqQBP

All permutations used in the previous section have been indeed members of $A_5 \subseteq S_5$ (the group of even permutations of 5 items). A_5 is the smallest non-Abelian simple group.

Definition 8.4 A group G is ***Abelian*** iff $ab = ba$ for all $a, b \in G$. A subgroup $H \subseteq G$ is ***normal*** iff $aH = Ha$ for all $a \in G$. A group G is ***simple*** iff it has no normal subgroups other than $\{1\}$ and G.

Let us restate the Barrington's result using the group language.

Theorem 8.7 Let G be a non-Abelian simple group and let $a \in G$, $a \neq 1$ be a non-identity element. Then any language L in NC_1 can be recognised by a sequence P of BP's over G of polynomial length such that $P(x) = a$ if $x \in L$ and $P(x) = 1$ if $x \notin L$.

Definition 8.5 We define 2-EqQBP to be the class of languages recognised exactly (in mode Eq) by sequences of width-2 oblivious QBP's of polynomial length.

Theorem 8.8 $\mathrm{NC}_1 \subseteq$ 2-EqQBP and w-EqQBP$\subseteq \mathrm{NC}_1$ (without proof here).

Proof. Recall that A_5 is the set of rotations of an icosahedron. Therefore $SO(3)$ (the group of rotations of $\mathbf{R}^3$, i.e. the 3×3 orthogonal matrices with determinant 1) contains a subgroup isomorphic to A_5.

There exists a well-known 2-to-1 mapping from $SU(2)$ (the group of 2×2 unitary matrices with determinant 1) to $SO(3)$. Recall the Bloch sphere representation of a qbit: if we neglect the unobservable global phase, a qbit $a|0\rangle + b|1\rangle$ where $|a|^2+|b|^2=1$ can be regarded as a point on the unit sphere with latitude θ and longitude φ, i.e. $(\cos\varphi\cos\theta, \sin\varphi\cos\theta, \sin\theta)$, where $a = \cos(\theta/2)$ and $b = e^{i\varphi}\sin(\theta/2)$.

Given this representation, an element of $SU(2)$ is equivalent to some rotation of the unit sphere. Recall the Pauli operators X, Y and Z and the rotation operators $R_x(\theta) = e^{-i\theta X/2}$, $R_y(\theta) = e^{-i\theta Y/2}$ and $R_z(\theta) = e^{-i\theta Z/2}$ rotating an angle θ around the x, y and z axes. This makes $SU(2)$ a double cover of $SO(3)$, where each element of $SO(3)$ corresponds to two elements $\pm U$ in $SU(2)$. The angles are halved by this mapping. Therefore $SU(2)$ has a subgroup which is a double cover of A_5.

One way to generate this subgroup is with $2\pi/5$ rotations around two adjacent vertices of an icosahedron. Since two such vertices are an angle $\tan^{-1}2$ apart, if one is pierced by the z axis and the other lies in the $x-z$ plane, we have

$$
\begin{aligned}
a &= R_z(2\pi/5) = \begin{pmatrix} e^{i\pi/5} & 0 \\ 0 & e^{-i\pi/5} \end{pmatrix}, \\
b &= R_y(\tan^{-1}2)\cdot a \cdot R_y(-\tan^{-1}2) \\
&= \frac{1}{\sqrt{5}} \begin{pmatrix} e^{i\pi/5}\tau + e^{-i\pi/5}\tau^{-1} & -2i\sin(\pi/5) \\ -2i\sin(\pi/5) & e^{-i\pi/5}\tau + e^{i\pi/5}\tau^{-1} \end{pmatrix},
\end{aligned}
$$

where $\tau = (1+\sqrt{5})/2$ is the golden ratio. Now consider the group element $c = aba$, this rotates the icosahedron by π around the midpoint of the edge connecting these two vertices. In $SU(2)$, this maps each of the eigenvectors of Y to the other times an overall phase. Taking these as the initial and final state $e_s = (|0\rangle + i|1\rangle)/\sqrt{2}$, $e_t = (|0\rangle - i|1\rangle)/\sqrt{2}$ we have

$$
\begin{aligned}
\| \langle e_s|c|e_t\rangle \|^2 &= 1, \\
\| \langle e_s|1|e_t\rangle \|^2 &= 0
\end{aligned}
$$

(because the two eigenvectors are orthogonal). Hence we have found a rotation c from the group such that we can measure with probability 1 whether or not it has been performed.

Now Theorem 8.7 tells us, that for any language $L \in \mathrm{NC}_1$ we can construct a polynomial length program over A_5 that yields the element equivalent to c if the input is in L and 1 otherwise. Mapping this language to $SU(2)$ gives a program which yields $\pm c$ or 1 and accepts with probability 1 or 0. □

Index

acceptance
  modes, 12, 90
advice, 32
  asking for, 32, 57
  length, 32
algorithm
  quantum
    Deutsch-Jozsa, 31
amplitude, 19, 28, 29, 38
  algebraic, 29, 37
  correction, 88, 91, 92
  negative, 88, 98
  rational, 29
array, 46, 55, 61, 85, 86

basis
  computational, 19, 21, 26, 32, 45, 72
  non-computational, 24
branching
  quantum, 60
branching program, 5, 14, 35, 87
  layered, 16
  oblivious, 105
  permutation, 105–108
  probabilistic, 10, 14
  quantum, 35, 37, 53
    bounded-degree, 71, 73, 74
    layered, 64–66
  reversible, 15, 105

circuit
  Boolean, 105
  quantum, 25, 47
coin
  fair flip, 13, 88–90, 99, 100, 103
    premature, 100
comparing
  numbers, 61
complexity
  classes, 14
    NC_1, 105–108
    classical, 13
    of quantum circuits, 25
    quantum, 31, 50, 57, 107
    quantum with advice, 32
  depth, 105, 107
  of a BP, 7
  of a circuit, 105
  of a PBP, 11
  of a QN, 49
  of a QTM, 30
  of a sequence of BP's, 8
  space, 7, 11, 30, 49, 50, 105
    bounded, 54
    computable, 53–55, 57
  time
    expected, 7, 11, 30, 49, 93
    maximal, 7, 11, 30, 49, 105, 106
computation
  history, 95, 100
  infinite, 37, 91
  iteration, 88, 91–93, 103
  of a BP, 6
  of a PBP, 11
  of a QN, 37, 38, 48

of a QTM, 30
path, 7, 14
consistent, 16, 80, 82
reversible, 15, 79, 96
condition
being well-formed, 29, 38
completeness, 23, 25
input/output, 61
normalisation, 20
parental, 15, 29, 36, 39, 43, 80, 82, 83, 85
unitarity, 36, 39, 54
configuration
of a QN, 37
of a QTM, 28
counter
error, 88, 89, 103, 104
time, 94

edge
back, 64
backward, 81
colour, 39
dashed, 6, 16
forward, 81
solid, 6, 16
encoding
of a BP, 8
modified, 85, 97
of a QN, 45, 55, 65, 75, 77, 85
extended, 66, 68, 97
of a QTM configuration, 30
entanglement, 21

family, 39
child, 60
decomposition, 40, 45, 47, 54, 56, 71
indivisible, 39, 41, 45
induced, 39, 40
parent, 59

function
constructible
space, 4, 8, 31
time, 4, 8, 31
creating, 93
transition
of a QN, 35, 37, 42, 43
of a QTM, 27, 29

game
Pebble, 95–98, 101, 103
gate
fan-in 2, 105–107
graph
acyclic, 4, 37, 80, 82, 87
component, 4, 37, 76, 77
connected, 4
diameter, 65
Euler, 79
monochromatic, 39, 40, 45, 47, 54, 55, 58, 59, 80
reachability, 65
sink, 4, 16
source, 4, 16, 17
group
$SO(3)$, 107, 108
$SU(2)$, 107, 108
commutator, 106
double cover, 108
non-Abelian, 107
normal subgroup, 107
permutation, 106, 107
simple, 107

halting
absolutely, 31, 49, 88, 93

inequality
Chebychev's, 12
information
erasure, 88
interference

quantum, 44, 45, 47, 58, 74, 88, 98
destructive, 99
interpretation, 89
of a PBP by a QN, 93
of a PBP by a QN (tradeoff), 103
of a PBP by an unreliable QBP, 89, 103, 104

language
acceptance, 7
five-cycle, 106
acceptance by a BP, 7
acceptance by a PBP, 11
acceptance by a QN, 48
acceptance by a QTM, 30
non-recursive, 8, 32
list
sorted, 61, 85, 86
unsorted, 61
loop
quantum, 61

machine
construction, 8, 46, 63–66, 68, 74, 77, 78, 85
Turing, 8, 14, 26, 46, 53
oblivious, 33, 58
quantum, 27, 44, 47
quantum with advice, 32
reversible, 33, 46, 56, 58
matrix
adjoint, 3
block, 3
block-diagonal, 3, 39, 41, 42
upper triangular, 72, 73
measurement, 23, 59, 88
deferred, 91
in the computational basis, 24
POVM, 23
projective, 23
model
nonuniform, 32, 44
uniform, 32, 44, 47

network, 87
quantum, 35, 53, 88
bounded-degree, 52, 70
connected, 76
equivalence, 48, 54
layered, 50, 51, 63, 66, 67
oblivious, 51, 68
notation
$O(f), o(f), \Omega(f), \Theta(f)$, 3
Dirac, 3
parameter _, 4

observable, 23
composition, 24
of a QN, 38, 42
of a QTM, 29
observation, 23
direct, 23
operator
approximation, 22, 47
bit flip, 21
controlled NOT, 22, 25
decomposition, 72–74
evolution, 21
of a QN, 38, 42, 43, 51, 52
of a QTM, 29
fanout, 26
Hadamard, 22, 25, 45, 76, 88, 89, 98, 100, 103
identity, 21, 64, 91
increment modulo, 61
parity, 26
Pauli, 21, 25, 108
phase flip, 21
quantum Fourier, 47
rotation, 22, 108

rotation of vectors, 70–72
Toffoli, 26
unitary, 21, 71, 72
universal set of, 22
XOR, 58–60, 98
overhead
space, 49, 54, 62
time, 49, 54, 62

phase
global, 20, 107
relative, 20
principle
Yao's, 31
probability
acceptance, 11, 30, 48, 90, 98, 100
squared, 89, 90, 93, 100, 103, 104
gain, 91, 93
problem
Reachability, 65–67
program
branching, 5

qbit, 19

random
choice
simple structure, 101
choice type, 13, 44, 88, 90, 94, 100, 103
reversibility
achieving, 35, 79
by back-tracking, 16, 67, 80, 83, 85, 86
by infinite iteration, 88, 93
by Pebble game, 96–98, 100, 101
local, 16
tradeoff, 101–103

search
back-tracking, 79
binary, 61, 85
exhaustive, 61, 85
sequence
nonuniform
of BP's, 8
of QN's, 46, 47
of BP's, 7
of circuits, 105
of QN's, 44
uniform
of BP's, 8, 105
of circuits, 105
of QN's, 46, 52, 54, 62, 63
simulation, 14, 33, 51, 53, 63
of a w-PBP by a circuit, 107
of a BP by a RBP (tradeoff), 102
of a BP by a RBP using back-tracking, 80, 85, 101, 102
of a BP by a RBP using Pebble game, 96, 97, 101, 102
of a BP by a TM with advice, 9
of a circuit by a 5-PBP, 106, 107
of a layered QN by an oblivious QN, 68
of a PBP by a QBP using Pebble game, 100, 101, 103
of a QBP by a layered QBP, 64, 65
of a QN by a bounded-degree QN, 70
of a QN by a connected QN, 76
of a QN by a layered QN, 63
of a QN by a QBP, 67, 85
of a QN by a QTM with advice, 58
of a QN by a regular one, 78

- of a QTM by a uniform QN, 53
- of a QTM with advice by a QN, 57
- of a TM by a BP, 9
- of a uniform BP by a TM, 10
- of a uniform QN by a QTM, 62
- of a unitary operator by a QBP, 73, 76
- of an unreliable QBP by a QN, 91, 103, 104

space
- Hilbert, 3

sphere
- Bloch, 20, 22, 107

state
- classical, 19, 29
- distinguishing, 25
- disturb, 23
- EPR-pair, 21
- hidden, 23
- mixed, 19
- pure, 19
- space, 19
- superposition, 19, 29, 33
- vector, 19

superposition, 19, 28, 38
- uniform, 22

tensor
- product, 21

theorem
- Savitch's, 67

transition
- type, 44–46, 54, 56, 57, 59, 69, 73, 75, 78

uncomputation, 56, 58, 60, 88, 96–99

variable
- decision, 45, 46, 51, 52, 56, 58–60, 67–69, 75, 77, 80, 86, 87

vector
- norm, 3

vertex
- bounded types, 13
- child, 4
- degree
  - bounded, 13
  - input, 4
  - output, 4, 12
- internal, 4
- interpolation, 80, 83, 85
- parent, 4
- rank, 81
- unreachable, 76

Bibliography

[AGK01] Ablayev F., Gainutdinova A., Karpinski M.: On Computational Power of Quantum Branching Programs. *Proc. FCT 2001.* Lecture Notes in Computer Science 2138:59–70, 2001.

[AMP02] Ablayev F., Moore C., Pollett C.: Quantum and Stochastic Branching Programs of Bounded Width. Los Alamos Preprint Archive, quant-ph/0201139 v1, 2002.

[Bar89] Barrington D. A.: Bounded-Width Polynomial-Size Branching Programs Recognise Exactly Those Languages in NC_1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[Bea89] Beame P.: A General Sequential Time-Space Tradeoff for Finding Unique Elements. *Proc. 21st Annual ACM Symposium on Theory of Computing*, 197–203, 1989.

[Ben89] Bennett C. H.: Time/space tradeoffs for reversible computation. *SIAM Journal of Computing*, 4(18):766-776, 1989.

[CS83] Carlson D. A., Savage J. E.: Size-Space Tradeoffs for Oblivious Computations. *Journal of Computer and System Sciences*, 26:65–81, 1983.

[GHMP01] Green F., Homer S., Moore C., Pollett C.: Counting, Fanout, and the Complexity of Quantum ACC. Los Alamos Preprint Archive, quant-ph/0106017 v1, 2001.

[Gru97] Gruska J.: Foundations of Computing. International Thompson Computer Press, 1997.

[MNT90] Mansour Y., Nisan N., Tiwari P.: The Computational Complexity of Universal Hashing. *Proc. 22nd Annual ACM Symposium on Theory of Computing*, 235–243, 1990.

[Moo99] Moore C.: Quantum Circuits: Fanout, Parity, and Counting. Los Alamos Preprint Archive, quant-ph/9903046, 1999.

[NHK00] Nakanishi M., Hamaguchi K., Kashiwabara T.: Ordered Quantum Branching Programs are more powerful than Ordered Probabilistic Programs under a bounded-width restriction. *Proc. 6th Annual International Conference on Computing and Combinatorics.* Lecture Notes in Computer Science 1858:467–476, 2000.

[NC00] Nielsen M. A., Chuang I. L.: Quantum Computation and Quantum Information. Cambridge University Press, 2000.

[Pap94] Papadimitriou C. H.: Computational Complexity. Addison-Wesley, 1994.

[Pre99] Prescill J.: Lecture Notes for Quantum Information and Computation Course. Available at `http://www.theory..caltech.edu/people/preskill/ph219/#lecture`.

[SS01] Sauerhof M., Sieling D.: Quantum Branching Programs: Simulations and Lower and Upper Bounds (Extended Abstract). Preliminary version of a paper, 2001.

[Sav70] Savitch W. J.: Relationship between nondeterministic and deterministic space complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sip97] Sipser M.: Introduction to the theory of Computation. PWS Publishing, 1997.

[Wat98] Watrous J.: Relationships Between Quantum and Classical Space-Bounded Complexity Classes. *IEEE Conference on Computational Complexity*, 210–227, 1998.

[Weg87] Wegener I.: The complexity of boolean functions. Wiley-Teubner, 414–441, 1987.

[Weg00] Wegener I.: Branching Programs and Binary Decision Diagrams, Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications, 1–44, 2000.

[Yao83] Yao A. C.: Lower Bounds by Probabilistic Arguments. *Proc. of the 24th IEEE Symp. on Foundations of Computer Science*, 420–428, 1983.