Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Informática

Instituto de Estudos em Computação e Informação Quânticas

Um Simulador Simbólico de Circuitos Quânticos

Alexandre de Andrade Barbosa

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Modelos Computacionais e Cognitivos

Bernardo Lula Júnior

Aércio Ferreira de Lima

(Orientadores)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

B238s Barbosa, Alexandre de Andrade
Um simulador simbólico de circuitos quânticos / Alexandre de Andrade Barbosa. — Campina Grande, 2007.
90f. : il.

Referências
Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande. Centro de Engenharia Elétrica e Informática.
Orientadores: Bernardo Lula Júnior e Aércio Ferreira de Lima.

1. Teoria da Computação 2. Computação Quântica 3. Circuitos Quânticos 4. Simulação de Circuitos Quânticos I. Título.

CDU 681.3

"UM SIMULADOR SIMBÓLICO DE CIRCUITOS QUÂNTICOS"

ALEXANDRE DE ANDRADE BARBOSA

DISSERTAÇÃO APROVADA EM 12.07.2007

PROF. BERNARDO LULA JÚNIOR, Dr.
Orientador

PROF. AÉRCIO FERREIRA DE LIMA, Dr.
Orientador

PROF. RUBENS VIANA RAMOS , Dr.
Examinador

PROF. BRUNO BARBOSA ALBERT, D.Sc
Examinador

CAMPINA GRANDE – PB

Resumo

A computação quântica é uma das mais promissoras tecnologias atuais, ela se baseia nos princípios da mecânica quântica para fornecer um paradigma computacional que promete um ganho expressivo de processamento. Com isso, problemas considerados intratáveis classicamente poderão ter soluções quânticas eficientes.

Como até então nenhuma máquina quântica efetiva foi criada, a simulação se tornou a alternativa mais viável para o estudo e o desenvolvimento da área. Em vista disso, é necessário disponibilizar um sistema computacional que permita uma descrição em nível apropriado de um algoritmo quântico e uma "máquina" para simular esta descrição. Na literatura relacionada o estudo da computação quântica é, em geral, realizado através da linguagem de circuitos quânticos. Nesta linguagem, um algoritmo é apresentado através de sua representação visual (sintaxe) aliada à descrição e manipulação simbólica do estado do sistema (semântica).

No presente trabalho é apresentado o processo de desenvolvimento de um *Computer Algebra System* (*CAS*) específico para o contexto de circuitos quânticos. O *CAS* foi implementado como uma extensão para o *Zeno*, tornando o simulador a única ferramenta do gênero a fornecer uma descrição completa da linguagem de circuitos.

A simulação simbólica incorporada ao *Zeno* permite que as descrições matemáticas do estado do sistema sejam criadas e manipuladas facilmente. Com isso, é possível simplificar ou exibir formas de representação alternativas para facilitar a compreensão e resolução do sistema investigado.

O uso conjunto da abordagem numérica e gráfica com a abordagem simbólica, facilita sobremaneira o entendimento de um algoritmo e a compreensão de sua descrição matemática. As funcionalidades atualmente fornecidas pelo *Zeno* agilizam o desenvolvimento de algoritmos, pois permitem que os usuários trabalhem de uma forma mais rápida e eficiente do que se executarem os cálculos à mão. Além disso, o *CAS* possibilita a apresentação de descrições fieis àquelas exibidas na literatura.

Abstract

Quantum computation is one of the most promising current technologies, it is based on the quantum mechanics principles to supply a computational paradigm that will bring an expressive processing power. Problems considered classically intractable will be efficiently solved using quantum computation.

Because no effective quantum machine has been implemented yet, the quantum computation simulation tools became the most viable alternative for the study and development of the area. According to this, it is necessary to provide a computational system that allows an appropriate description of a quantum algorithm and a "machine" to simulate this description. In related literature the study of quantum computation is, in general, presented using the quantum circuits language. The understanding of a quantum algorithm is presented through the graphical representation of the circuit (syntax) allied to the symbolic description and manipulation of the system's state (semantics).

In this work it is presented the development process of a Computer Algebra System (CAS) specific for the context of quantum circuits. The CAS was implemented as an extension for Zeno, allowing the simulator to become the only tool that supplies a complete description of the circuits language.

The symbolic simulation incorporated to Zeno allows the mathematical descriptions of the system's state to be created and manipulated easily. In this way, it is possible to simplify or to show alternative representation forms that facilitate the understanding and resolution of the investigated problem.

The use of the numerical and graphical utilities allied to the symbolic simulation of quantum circuits, induces and improves the understanding of quantum algorithms and its associated mathematical description. The currently supported functionalities allow the users to work in a faster and more efficient way than making this calculations by hand. Moreover, the CAS also allows a faithful description of the system computation as the descriptions observed in related literature.

Agradecimentos

A Deus por me dar saúde e força para superar todas as dificuldades e por todas as pessoas que colocou em meu caminho.

Aos meus pais Ailton e Ellen, pelo incentivo, apoio, carinho, atenção, compreensão e por outros tantos motivos.

Ao meu grande amor, minha namorada Lívia, pela compreensão e carinho nas inúmeras vezes que não pude lhe dar a atenção merecida.

À minha querida irmã Christiane por me incentivar no estudo de uma segunda língua, no caso a língua inglesa tão importante para o curso que escolhi.

A Cida pelo carinho e paciência.

Aos companheiros de apErtamento Milena e Fred, com os quais tantas contas dividi.

Aos orientadores Bernardo e Aércio, por acreditarem em minha competência para realizar o trabalho e pelos valiosos conselhos.

Ao pessoal do IQuanta, principalmente a Alessandra e Cheyenne, pela paciência e atenção.

Aos professores pelo aprendizado proporcionado.

A todos os amigos que fiz durante curso e outros com os quais fortaleci ainda mais minha amizade.

A George Lucas, Steven Spielberg, Stan Lee, Steve Ditko, Tolkien e tantos outros que criaram as inúmeras obras de ficção que tanto gosto, extremamente importantes para que pudesse relaxar após cada dia de trabalho, permitindo que ficasse renovado para o dia seguinte.

A todos que de maneira direta ou indireta contribuíram para a conclusão desta fase de minha vida.

À CAPES pelo apoio financeiro na forma de bolsa de mestrado.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste trabalho, é apresentado o processo de desenvolvimento da extensão do simulador de circuitos quânticos *Zeno*. A extensão adicionou ao simulador um Sistema de Álgebra Computacional (SAC), ou *Computer Algebra System* (*CAS*), específico para o contexto de circuitos quânticos. Com isso o simulador passa a fornecer ao usuário a representação gráfica do circuito aliada a descrição e manipulação simbólica do estado do sistema, características não oferecidas de maneira conjunta em nenhuma outra ferramenta do gênero. Isto posto, pode-se afirmar que o simulador *Zeno* fornece uma descrição completa da linguagem de circuitos criada por Deutsch [Deu89].

Neste capítulo, são realizadas considerações sobre os limites da computação clássica. Além disso, são apresentadas, de maneira sucinta, a importância, os problemas e as vantagens relacionadas à computação quântica. Por fim, são exibidos os objetivos, a relevância e a estrutura do trabalho.

1.1 Os limites da computação clássica

Atualmente o aumento da capacidade de processamento dos computadores vem ocorrendo de acordo com a Lei de Moore (*Mooret's Law*) [Moo65], assim chamada devido a observação realizada por Gordon Moore, em 1965. Moore afirmou que a cada doze meses a capacidade de processamento dos computadores dobrava, anos depois a observação foi reafirmada, porém desta vez para um período de vinte e quatro meses. A Lei de Moore vem se mostrando correta a mais de quarenta anos, todavia devido à limites físicos, es-

pecialistas acreditam que ela será válida somente por mais um curto período [Llo00; Fra05], caso nenhuma nova tecnologia venha substituir a atual.

O motivo da limitação da atual tecnologia ocorre por esta se basear na miniaturização de componentes, os chips são atualmente desenvolvidos em escala de aproximadamente 65 ηm, e espera-se que até 2010 a escala seja de 32 ηm [Boh02; Boh05]. Segundo Louis de Broglie as partículas deixam de se comportar como previsto pela física clássica, obedecendo somente aos princípios da mecânica quântica abaixo de 20 ηm [Bro24]. Assim, a criação de uma nova tecnologia baseada na mecânica quântica pode garantir a evolução da computação, no que se refere ao seu poder de processamento. Entre as tecnologias que estão sendo pesquisadas a computação quântica [NC00] se mostra como uma das mais promissoras, pois se baseia nos princípios da mecânica quântica para fornecer um paradigma computacional que promete um ganho expressivo de processamento.

1.2 A computação quântica

O interesse relacionado à computação quântica iniciou-se a partir do trabalho de Feynman [Fey82], onde foi observado que nenhum sistema clássico pode simular um sistema quântico de maneira eficiente. A partir desta observação, Feynman propôs que apenas um sistema quântico pode simular eficientemente outro sistema quântico. Nascia assim a proposta de que computadores quânticos poderiam ser utilizados para modelar sistemas da mecânica quântica.

Posteriormente, David Deutsch indaga-se se computadores quânticos poderiam realizar computações de maneira mais eficiente do que os clássicos. Buscando respostas para este questionamento ele cria o modelo de computador quântico universal e implementa um algoritmo quântico (*Deutsch's two bit problem*) mais eficiente do que qualquer algoritmo clássico [Deu85]. Porém, o trabalho só passa a ser valorizado quando Deutsch cria a linguagem de circuitos quânticos [Deu89], que passa a ser amplamente aceita pela comunidade por ser bastante similar a linguagem de circuitos clássica.

Outros algoritmos quânticos que ilustram as vantagens da computação quântica foram criados, dois exemplos são o algoritmo de Bernstein e Vazirani [BV93] e o algoritmo de Simon [Sim94], mas somente com o trabalho de Peter Shor [Sho94], o qual realiza fatoração

de números inteiros em tempo polinomial, é que a computação quântica deixou de ser vista apenas como uma curiosidade acadêmica.

O algoritmo de Shor, aliado à enorme importância adquirida pelos sistemas de criptografia de chave pública, estimulou o estudo da computação quântica tanto na direção da construção de computadores quânticos, quanto no desenvolvimento de algoritmos. Os algoritmos de Grover [Gro96], para busca em bases de dados desordenadas em O(n) e de Jozsa [Joz97], que exibe um algoritmo de fatoração alternativo, são exemplos de outros algoritmos quânticos importantes que surgiram após o trabalho de Shor.

Com relação à construção de máquinas quânticas, alguns projetos desenvolveram sistemas quânticos com poucos qubits, dentre os quais podem ser citados [JM98], [JMH98], [CGK98] e recentemente [D-W07]. Contudo, os sistemas implementados não são suficientes para realizar computações úteis, sendo assim, pode-se afirmar que ainda não existe um computador quântico efetivo.

Existem sérias dificuldades relacionadas à implementação de uma máquina quântica, algumas destas são a escalabilidade e a descoerência. A escalabilidade se refere à quantidade de qubits utilizados nas implementações, as quais, até o momento, utilizam poucos qubits. A descoerência diz respeito à dificuldade existente em se manter um sistema quântico isolado do ambiente, pois, a interação com o ambiente leva à corrupção da informação quântica.

Devido às dificuldades relacionadas à implementação de hardware quântico, ferramentas de simulação vêm sendo utilizadas no desenvolvimento da computação quântica. O trabalho mais completo sobre simuladores quânticos produzido até o momento foi desenvolvido por Julia Wallace [Wal99]. Neste trabalho, são descritas diversas ferramentas relacionadas à simulação da computação quântica, tais como: simuladores de circuitos, simuladores de máquinas de Turing, linguagens de programação. Um trabalho mais recente, porém menos abrangente, pois trata apenas de simuladores de circuitos quânticos, foi desenvolvido por Gustavo Eulálio e Bernardo Lula [CL03]. Neste trabalho, diversos simuladores de circuitos quânticos são descritos e comparados. As ferramentas exibidas nos dois trabalhos citados, além de outras encontradas através de pesquisas realizadas pelo autor, foram analisadas e comparadas. As considerações apresentadas no presente trabalho tratam apenas da simulação de circuitos quânticos, a não ser que alguma observação seja realizada.

1.3 Objetivos

O trabalho aqui apresentado teve como objetivo desenvolver um *CAS*, específico para o contexto de circuitos quânticos, como extensão para o simulador *Zeno*. A extensão do simulador possibilita que a representação gráfica de um circuito seja visualizada em conjunto com a descrição simbólica do estado do sistema. O *CAS* permite ainda a manipulação das expressões matemáticas que descrevem as transformações de estado existentes em um circuito, em busca de representações equivalentes mais simples ou compreensíveis.

O primeiro algoritmo quântico que utilizou efetivamente operações quânticas foi apresentado em 1985 por David Deutsch. O algoritmo de Deutsch, como é atualmente chamado, determina se uma função $f : \{0,1\} \rightarrow \{0,1\}$, é balanceada, $f(0) \neq f(1)$, ou constante, $f(0) = f(1)$, em uma única execução, enquanto no caso clássico é necessário realizar duas execuções. O Exemplo 1.1 apresenta o algoritmo de Deutsch, ilustrando o uso conjunto da representação gráfica do circuito e da descrição simbólica do estado do sistema.

Exemplo 1.1. *A representação gráfica do circuito isolada não possibilita uma compreensão completa do algoritmo.*

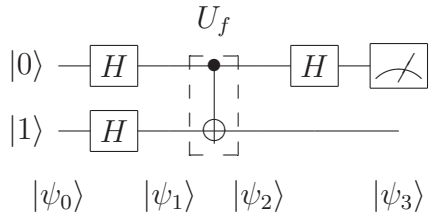


Figura 1.1: Circuito implementando o algoritmo de Deutsch para a função soma módulo 2.

Cada estado intermediário possui diversas representações equivalentes, por exemplo, o estado inicial $|\psi_0\rangle$ pode ser representado através de qualquer uma das expressões exibidas a seguir:

$$|\psi_0\rangle = |0\rangle|1\rangle = |01\rangle = |0\rangle \otimes |1\rangle \tag{1.1}$$

Uma forma de representação pode ser mais agradável para compreender o estado do sistema, enquanto outra pode facilitar a execução de determinada manipulação.

Para alcançar o estado $|\psi_1\rangle$ são aplicados operadores Hadamard aos respectivos qubits de entrada, obtendo-se assim todos os possíveis valores de entrada para a função. Este estado pode ser descrito pelas expressões a seguir:

$$\begin{array}{lll} |\psi_1\rangle & = (H \otimes H)(|0\rangle \otimes |1\rangle) & = H|0\rangle H|1\rangle \\ & = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) & = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ & = \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} (-1)^{i_1} |i_0 i_1> & \end{array} \tag{1.2}$$

Diversas manipulações podem ser aplicadas em cada representação intermediária até que a forma desejada seja obtida. Em $|\psi_2\rangle$ todas as saídas da função foram computadas, desta forma, pode-se descrever o estado como:

$$\begin{array}{lll} |\psi_2\rangle & = U_f \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) & = (X^1)\frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ & = \frac{1}{2}(|00\rangle - |01\rangle + |11\rangle - |10\rangle) & = \sum_{i_0=0}^{1} \sum_{i_1=0}^{1} (-1)^{f(i_0 i_1)} |i_0 i_1> \end{array} \tag{1.3}$$

Combinando a execução de passos de simulação com a manipulação das expressões, pode-se compreender um algoritmo de maneira mais simples do que através de uma descrição exclusivamente numérica.

Finalmente, no estado $|\psi_3\rangle$ é aplicado novamente o operador Hadamard sobre o primeiro qubit, obtendo-se uma descrição equivalente as apresentadas a seguir:

$$\begin{array}{lll} |\psi_3\rangle & = (H \otimes I)\frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle) & = \frac{1}{\sqrt{2}}(-|1\rangle \otimes (|0\rangle - |1\rangle)) \\ & = \frac{1}{2}H(|0\rangle - |1\rangle) \otimes I(|0\rangle - |1\rangle) & = \frac{1}{\sqrt{2}}(-|10\rangle + |11\rangle) \end{array} \tag{1.4}$$

Realizando a medição sobre o primeiro qubit identifica-se se a função computada é balanceada (se o estado corresponder ao qubit $|1\rangle$), ou constante (se o estado corresponder ao qubit $|0\rangle$). Assim, pode-se concluir que a função computada no exemplo é balanceada.

A escolha do *Zeno* como ferramenta a ser estendida ocorreu devido a uma série de motivos. Um destes motivos é o fato de que o *Zeno* representa o estado da arte no que se refere a simulação de circuitos quânticos, as funcionalidades do simulador englobam o conjunto de funcionalidades existente em outras ferramentas do gênero. Uma outra vantagem é que o simulador foi desenvolvido na Universidade Federal de Campina Grande (UFCG), sendo assim, tem-se fácil acesso aos desenvolvedores da mesma e a um bom número de usuários.

1.4 Relevância

Como é mostrado na literatura sobre computação quântica, o entendimento de um algoritmo quântico na linguagem dos circuitos passa pela representação gráfica do circuito (sintaxe), aliada à descrição e manipulação simbólica do estado do sistema (semântica). Fica claro então, que a união destas abordagens é a melhor maneira para compreender, investigar e desenvolver os mais variados aspectos da área.

Além de apoiar o ensino, a extensão também é útil em contextos de pesquisa onde a simplificação e as transformações de expressões podem facilitar a visualização dos diversos "caminhos" possíveis na busca por uma descrição mais compreensível do tema pesquisado.

Sabe-se que os *CAS* vêm apoiando com sucesso o ensino e a pesquisa em diversas áreas: o *Maple*, por exemplo, foi utilizado no ensino de mecânica quântica, na Universidade de Lethbridge no Canadá [Rou99]. Já as ferramentas de simulação usadas na computação quântica têm mostrado grande relevância, pois facilitam a distribuição de experimentos e ilustram diversos conceitos relacionados. Assim, nada mais natural do que unir o melhor dos dois mundos, como realizado no presente trabalho.

Um outro benefício relacionado à extensão do *Zeno* foi a divulgação e o reconhecimento em nível nacional e internacional do grupo de pesquisas em computação e informação quântica da UFCG.

1.5 Estrutura da dissertação

O presente trabalho de dissertação está estruturado da seguinte maneira:

• Capítulo 2 - apresenta os conceitos fundamentais para compreensão do trabalho;

• Capítulo 3 - exibe considerações relacionadas à simulação de circuitos quânticos, além de descrever algumas ferramentas para simulação destes circuitos;

• Capítulo 4 - descreve o simulador Zeno, ilustrando as funcionalidades originalmente suportadas pelo mesmo;

• Capítulo 5 - apresenta a extensão do simulador Zeno, mostrando as funcionalidades relacionadas à exibição e a manipulação de expressões;

• Capítulo 6 - descreve os resultados obtidos no trabalho;

• Capítulo 7 expõe as conclusões relacionadas ao trabalho, além de exibir alguns temas que poderão ser explorados posteriormente.

Capítulo 2

Fundamentação Teórica

Neste capítulo, serão apresentados os conceitos relacionados à linguagem de circuitos quânticos e aos Sistemas da Álgebra Computacional, os quais formam a base teórica necessária ao entendimento do trabalho.

2.1 A linguagem de circuitos quânticos

Apenas os conceitos necessários à compreensão da linguagem de circuitos quânticos serão apresentados, quanto à uma introdução mais detalhada da computação quântica recomenda-se [NC00] ou [PLCM04].

2.1.1 Representação da informação

Os estados e portas quânticas correspondem respectivamente a vetores e operadores lineares, a notação comumente utilizada na representação das expressões de descrição do estado do sistema é a chamada de notação de Dirac ou *braket*. Sendo essa notação um meio útil para facilitar e agilizar a execução de operações no contexto da computação quântica, caso não esteja familiarizado com a notação consulte o Apêndice A.

O bit quântico (qubit)

A unidade básica de informação para a computação clássica é o *binary digit* (*bit*), que pode assumir os valores 0 (zero) ou 1 (um), já na computação quântica o conceito análogo é o

quantum bit (qubit), que é representado matematicamente por um vetor complexo unitário em um espaço de Hilbert. Sendo assim os valores 0 e 1 são substituídos pelos vetores $|0\rangle = [1,0]^T$ e $|1\rangle = [0,1]^T$, além disso, um qubit pode também ser uma combinação linear destes, neste caso, diz-se que ele está em *superposição*.

Registradores quânticos

Um conjunto de *bits* na computação clássica é chamado de registrador, do mesmo modo, na computação quântica um conjunto de qubits é denominado de registrador, porém este freqüentemente é chamado de *estado do sistema* ou *estado*. Enquanto um registrador clássico é capaz de armazenar 2^n números distintos por vez, um registrador quântico de *n* qubits pode armazenar 2^n números distintos simultaneamente.

Para representar um registrador quântico utiliza-se o produto tensorial, por exemplo para os qubits $|\psi\rangle$ e $|\phi\rangle$ representados por

$$|\psi\rangle = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_m \end{bmatrix} \text{ e } |\phi\rangle = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}$$

pode-se definir, de maneira simples, o produto tensorial como a seguir:

$$|\psi\rangle \otimes |\phi\rangle = |\psi\rangle\,|\phi\rangle = |\psi\phi\rangle = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_m \end{bmatrix} \otimes \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} \psi_1\phi_1 \\ \vdots \\ \psi_1\phi_n \\ \vdots \\ \psi_m\phi_1 \\ \vdots \\ \psi_m\phi_n \end{bmatrix} \tag{2.1}$$

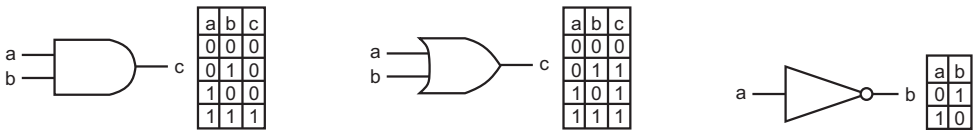
Na representação de qubits usualmente utiliza-se a base binária, mas assim como em um registrador clássico, pode-se utilizar a base decimal, por exemplo $|00\rangle = |0\rangle$, $|01\rangle = |1\rangle$, $|10\rangle = |2\rangle$, $|11\rangle = |3\rangle$.

2.1.2 Processamento da informação

Assim como ocorre em um circuito clássico, um circuito quântico é composto por um conjunto de portas, porém, naquele as portas representam operações lógicas, enquanto neste as portas são operadores lineares.

Portas quânticas de um qubit

As portas clássicas são representadas graficamente através de "modelos" diferentes. Uma porta clássica executa uma operação lógica sobre um ou mais *bits* de entrada gerando um *bit* de saída. A Figura 2.1 exibe alguns exemplos de portas clássicas e ilustra sua atuação.



(a) Ação da porta *AND* (b) Ação da porta *OR* (c) Ação da porta *NOT*

Figura 2.1: Exemplos de operações lógicas

Na computação quântica o processamento da informação ocorre de maneira similar à clássica. A representação gráfica destas portas faz uso de "caixas", que são diferenciadas por rótulos identificadores. Alguns exemplos de portas quânticas são exibidos na Figura 2.2.

$$-\boxed{X}- = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, -\boxed{Y}- = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, -\boxed{Z}- = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, -\boxed{H}- = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figura 2.2: Exemplos de portas quânticas

A porta *NOT* clássica nega o valor do *bit* de entrada, seu equivalente quântico, representado pela porta X, realiza a mesma operação sobre o qubit de entrada. Pode-se visualizar a ação de X a seguir:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \quad , \quad X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.2}$$

Outro exemplo de porta quântica de um qubit é a porta H, denominada Hadamard. Esta porta assim como várias outras portas quânticas não possui uma porta clássica equivalente. A aplicação de H a um estado da base canônica gera uma superposição, já a aplicação deste mesmo operador a uma superposição igualmente ponderada gera um estado da base canônica. Pode-se observar alguns exemplos de aplicação do operador Hadamard a seguir:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.3}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.4}$$

$$H\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.5}$$

Da mesma maneira que o produto tensorial é adotado na representação de estados, ele pode ser utilizado na descrição de operadores, por exemplo:

$$(H \otimes H) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{2.6}$$

com isso, este novo operador só pode ser aplicado sobre um estado de mesma aridade.

Portas quânticas de n-qubits

Apesar do conjunto de portas quânticas de um qubit ser infinito, pois existem infinitos operadores unitários, este conjunto não é universal. É necessário utilizar operadores que atuem sobre mais de um qubit para que seja possível implementar qualquer tipo de operação.

As portas controladas são exemplos de portas que atuam sobre mais de um qubit, elas executam uma operação sobre um qubit alvo, dependendo do estado do(s) qubit(s) de controle. Um controle é representado por um dos símbolos • ou ○, ele é ativado quando o estado do qubit corresponde a $|1\rangle$, para o primeiro símbolo, e $|0\rangle$, para o segundo símbolo. Como os estados de entrada de uma porta quântica podem estar em superposição, então deve-se considerar a aplicação da porta controlada sobre todos os estados da superposição em que ela atua. A Figura 2.3 exibe uma porta *U-controlada*.

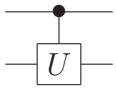


Figura 2.3: Porta U-controlada.

Uma das portas controladas mais utilizadas no contexto de circuitos quânticos é a porta *NOT-controlada* ou *CNOT*, esta nega o qubit alvo caso o qubit de controle seja ativado. A representação gráfica de *CNOT* e exemplos de sua computação são exibidos na Figura 2.4.

Figura 2.4: Representações da porta *CNOT*.

O conjunto formado por portas quânticas de um qubit mais a porta *CNOT* é universal [NC00], ou seja, todas as operações unitárias podem ser implementadas utilizando apenas estas portas.

Circuitos quânticos

Apesar da linguagem de circuitos quânticos ser similar à linguagem de circuitos clássicos, existem algumas notações e convenções que devem ser apresentadas para que ocorra um melhor entendimento. Tal como em [PLCM04] utiliza-se o circuito exibido na Figura 2.5 para apresentar as notações e convenções da linguagem de circuitos quânticos.

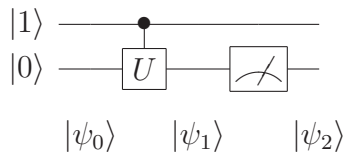


Figura 2.5: Circuito com porta quântica U-controlada e medição.

Nos circuitos clássicos e quânticos o lado esquerdo corresponde a entrada e o lado direito à saída, ressaltado que apenas no caso quântico estados em superposição podem estar

presentes no registrador. No circuito adotado para o exemplo, pode-se verificar que a entrada corresponde ao estado $|1\rangle \otimes |0\rangle$.

Um registrador quântico evolui à medida que portas são aplicadas aos respectivos qubits de entrada. As linhas horizontais representam a evolução dos qubits, assim pode-se determinar quais operadores serão aplicados a um estado. Em um circuito quântico, diferentemente de um circuito clássico, linhas verticais não representam uma cópia, elas indicam que determinado(s) qubit(s) controlam a aplicação de uma porta.

Freqüentemente são utilizados rótulos para representar os estados intermediários de uma computação, no circuito exibido na Figura 2.5 estes rótulos são $|\psi_0\rangle$, $|\psi_1\rangle$ e $|\psi_2\rangle$. A última porta aplicada ao estado $|0\rangle$ corresponde a representação padrão de uma medição.

2.2 Algebra Computacional (*CA*)

Expressões matemáticas possuem grande importância para as ciências exatas, tais como: física, computação, engenharias. Isso ocorre pois estas ciências estudam problemas que podem ser descritos através destas expressões. Assim, pode-se afirmar que a manipulação de uma expressão em busca de uma representação mais simples ou compreensível irá facilitar o entendimento do problema representado.

A manipulação de expressões é uma tarefa lenta e sujeita a erros, por isso a computação vem sendo usada na obtenção de meios automáticos para execução desta atividade. O estudo de manipulações matemáticas está inserido na área de Álgebra Computacional ou Computação Algébrica (*Computer Algebra - CA*), sendo este termo utilizado para definir computações numéricas ou simbólicas. Os pesquisadores desta área, que esta associada à matemática e à computação, têm como objetivo desenvolver algoritmos e aplicações que realizem manipulações e análises de expressões matemáticas.

2.2.1 Um breve histórico

Como descrito em [Dav05], os primeiros trabalhos relacionados à *CA*, desenvolvidos nas décadas de 50 e 60, possuíam como tema a diferenciação automática. Os primeiros sistemas criados neste período eram bastante complicados e trabalhavam com áreas restritas, posteriormente quando sistemas mais abrangentes foram criados, eles eram bem mais fáceis de

usar do que os anteriores. Durante as décadas de 70 e 80, muitos avanços significativos ocorreram, com isso os sistemas passaram a ter muitas características similares as existentes nos sistemas contemporâneos.

Atualmente, os pesquisadores da área procuram alternativas para: melhorar a interação com o usuário, em [Fat04], por exemplo, busca-se criar um mecanismo que possibilite a execução de cálculos utilizando comandos de voz; possibilitar a comunicação entre *CASs*, característica desejável para que os sistemas possam realizar solicitações, seja devido à eficiência ou à inexistência de alguns algoritmos específicos em sistemas de propósito geral.

2.2.2 Manipulações e representações simbólicas

Muitos usuários possuem a idéia de que apenas cálculos simbólicos são adequados para o uso da *CA*. A manipulação de expressões numéricas ou simbólicas pode ser executada com o apoio da *CA*, desde que o usuário realize as solicitações adequadas.

Os algoritmos de manipulação da *CA* exibem preferencialmente formas de representação que mantém a exatidão do valor que se deseja apresentar, assim não é necessário realizar aproximações, tal como ocorre freqüentemente em um cálculo exclusivamente numérico. Além disso, as representações obtidas com o uso da *CA* são, em geral, mais simples de compreender do que sua forma numérica equivalente. Na Tabela 2.1 são apresentados exemplos de computações numéricas e suas formas equivalentes na *CA*.

Tabela 2.1: Exemplos de computações numéricas e sua forma equivalente na *CA*.

Valor	Representação numérica equivalente	Representação equivalente na *CA*
$\frac{2}{6}$	$0.3333...$	$\frac{1}{3}$
π	$3.1416...$	π
$6+7$	13	13
$2x+x$	–	$3x$
$\sqrt{2}$	$1.4142135...$	$\sqrt{2}$
$\frac{2}{6}+\sqrt{2}$	$1.7475...$	$\frac{1}{3}+\sqrt{2}$
$2x+4y$	–	$2(x+2y)$
$2\times\sqrt{2}/6$	$0.4714...$	$\sqrt{2}/3$

2.2.3 Sistemas de Álgebra Computacional (*CAS*)

Os *softwares* criados para *CA* são chamados de Sistemas da Álgebra Computacional (*Computer Algebra Systems - CASs*), estes podem ser classificados como de propósito geral ou de propósito específico dependendo de seu escopo de atuação, no entanto, estes sistemas não são classificados de maneira unânime na literatura. A subjetividade existente nesta classificação se deve as diferentes visões de mundo, sob uma ótica um sistema pode ter propósito geral e sob outra ótica pode-se classificar o mesmo sistema como de propósito específico. Contudo, seja o sistema de propósito geral ou específico, o objetivo final é o mesmo: facilitar a execução de manipulações sobre expressões simbólicas, minimizando a possibilidade de erro humano.

Em todos os sistemas a execução de manipulações só ocorre após a solicitação do usuário, esta estratégia é adequada, pois o usuário é o mais indicado para realizar ponderações sobre qual manipulação é mais indicada. Os *CASs* geralmente possuem linguagens de programação que possibilitam aos usuários desenvolver seus próprios algoritmos de manipulação.

A forma de interação mais comum se baseia no uso de *prompt de comando*, através do qual o usuário realiza suas solicitações. Muitos sistemas também fornecem *interfaces gráficas com o usuário* (*GUI*), as quais facilitam a interação e minimizam o tempo de aprendizado.

Alguns dos principais *CASs* existentes são: *Axiom*, *Derive*, *Macsyma/Maxima*, *Maple*, *Mathematica*, *MatLab*, *MuPAD* e *Reduce*.

Maple

O *Maple* foi desenvolvido inicialmente como um projeto de pesquisa na Universidade Waterloo em 1980 no Canadá. Nesta época, os sistemas mais populares eram *Macsyma* e *Reduce*, ambos desenvolvidos em *LISP*. Tais sistemas consumiam uma grande quantidade de recursos de modo que só eram utilizados em grandes *mainframes*, o que os tornava inacessíveis a uma grande parcela de usuários. Diante deste cenário, um dos objetivos do projeto de desenvolvimento do *Maple* era a popularização da *CA*, portanto, para que o sistema pudesse rodar em computadores pessoais figuravam entre seus requisitos principais a portabilidade e a velocidade.

A primeira versão do sistema, desenvolvida em C, possuía um número limitado de funcionalidades, apesar de diversas versões terem sido implementadas para fins de pesquisa, só em 1985 é que foi criada a primeira versão comercial do sistema. Em 1987 o sistema foi incorporado à *Waterloo Maple Software* (*Maplesoft*), onde continua a ser desenvolvido. Desde então o *Maple* vem sendo útil em inúmeras áreas tornando-se um dos líderes de mercado.

As funcionalidades existentes podem ser acessadas pelos comandos fornecidos através de um *prompt* ou por botões e menus, como pode ser observado na Figura 2.6. O *Maple* possui linguagem de programação própria através da qual os usuários podem desenvolver seus algoritmos de manipulação.

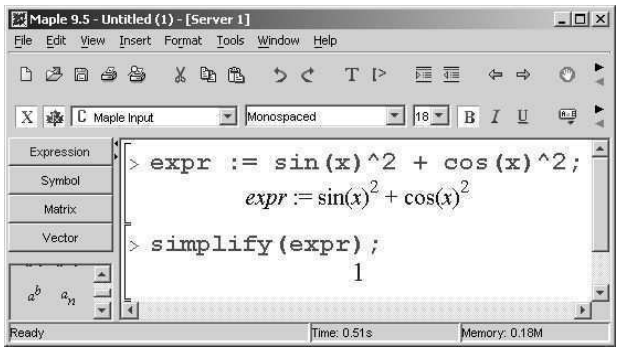


Figura 2.6: Interface do *Maple* versão 9.5

Uma das principais características deste *software* é a grande quantidade de pacotes de extensão, chamados de *Maple Worksheets*, os quais são disponibilizados gratuitamente. Estes incluem instruções de execução e tutoriais sobre o assunto de que tratam e podem ser lidos e executados somente no próprio sistema.

O *Maple* possui uma vasta documentação, a qual inclui livros, cursos de treinamento, manuais, *eBooks*, vídeos. Além disso, existem diversos produtos que podem ser acoplados ao sistema expandindo ainda mais seu poder.

Mathematica

O sistema *Mathematica* é hoje o mais popular *CAS* existente, foi inicialmente desenvolvido por Stephen Wolfram em 1986, tendo sua primeira versão lançada em 1988. Atualmente a empresa *Wolfram Research Inc.* [Wol05] desenvolve e comercializa o sistema.

Assim como os outros *CASs*, o *Mathematica* fornece uma linguagem de programação

através da qual os usuários podem manipular as expressões matemáticas desejadas. O usuário pode utilizar as funcionalidades do sistema realizando chamadas aos comandos através da *interface* padrão fornecida ou usando um *frontend* de terceiros, como por exemplo *JMath* [Rad02]. A interface padrão do sistema é exibida na Figura 2.7.

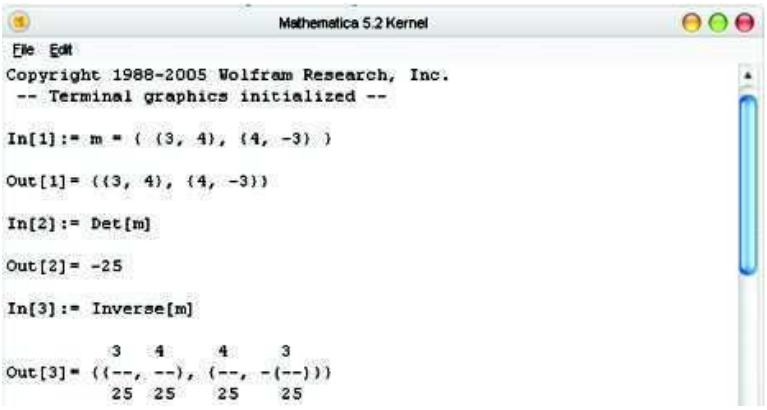


Figura 2.7: Interface do *Mathematica* versão 5.2

A comunicação entre programas de terceiros e o *Mathematica* pode ocorrer através do protocolo de comunicação *MathLink* [Gay05]. A *Wolfram Research* distribui gratuitamente kits de desenvolvimento em C ou Java para realizar a ligação entre seu sistema algébrico e qualquer outro *software* criado.

Os pacotes do *Mathematica* são denominados *Mathematica Notebooks*, eles podem ser desenvolvidos pelos usuários do sistema, são distribuídos gratuitamente e podem ser lidos através do *MathReader* ou lidos e executados no *Mathematica*.

Além dos *Mathematica Notebooks* existem inúmeros outros documentos relacionados ao sistema, tais como: artigos, manuais de utilização, tutoriais, livros, entre outros. Além destes, há ainda diversos outros documentos criados pelos usuários do sistema, onde pode ser encontrada uma grande variedade de informações.

2.2.4 Vantagens e limitações da *CA*

Pode-se indagar se os computadores são adequados para executar manipulações simbólicas de expressões matemáticas, uma vez que tais máquinas foram projetadas para realizar cálculos numéricos. Contudo, o sucesso obtido pelos *CASs*, os quais muitas vezes superam o usuário na execução deste tipo de atividade, tem mostrado que os computadores podem ser

utilizados para este fim.

Os *CASs* podem automatizar a execução de cálculos minimizando os erros humanos nesta atividade, visto que computadores não cometem erros se forem corretamente programados. Uma outra vantagem é que manipulações que poderiam exigir um tempo considerável, caso fossem executadas a mão, são realizadas em segundos ou milésimos de segundo.

Uma desvantagem associada aos *CAS* está relacionada a quantidade de tempo e memória utilizados em uma manipulação. Muitas vezes a quantidade de recursos necessários supera o processamento existente na execução de um cálculo numérico equivalente.

Em [Wes99] é apresentada uma comparação entre *CASs*, onde são exibidos 542 problemas que ilustram algumas limitações dos sistemas. Além das limitações relacionadas à resolução dos problemas, são exibidas algumas considerações gerais sobre, por exemplo, a documentação e as formas de interação destes sistemas. Com relação a documentação, é citado que, em geral, os tutoriais e manuais oferecidos contém apenas a lista dos comandos disponíveis no sistema. Devido a isso, o usuário é obrigado a realizar buscas por sinônimos, ou muitas vezes abreviações de sinônimos, das ações que ele deseja executar. Já quanto à interação, é dito que, para utilizar os sistemas de maneira satisfatória é necessário aprender a sintaxe de um boa quantidade de comandos, com isso, a utilização de recursos avançados destes sistemas só ocorre após um longo período de aprendizado.

Apesar das dificuldades e problemas citados, as vantagens relacionadas à utilização de *CASs* justificam seu uso, por isso, cada vez mais a *CA* vem sendo utilizada como ferramenta de apoio ao ensino e à pesquisa. Ela tem se mostrado uma boa alternativa na investigação de problemas onde muitos cálculos são necessários.

Capítulo 3

Simuladores de circuitos quânticos

Neste capítulo, serão apresentados os motivos que tornam a simulação importante de modo geral e para a computação quântica. Além disso, serão descritas algumas ferramentas relacionadas ao contexto de simulação de circuitos quânticos, sendo estas divididas em duas categorias: simuladores simbólicos e simuladores universais.

3.1 A importância da simulação

Em diversas áreas do conhecimento humano a simulação desempenha um importante papel, seja no estudo ou no desenvolvimentos de tais áreas. O uso da simulação acarreta um grande número de vantagens, a velocidade na obtenção das simplificações é o melhor exemplo dos benefícios obtidos.

Simuladores são utilizados, por exemplo, como ferramenta de apoio ao ensino. Para o ensino de matérias relacionadas à computação, existem ferramentas para simulação de arquiteturas de computadores clássicos, autômatos finitos e máquinas de Turing. Alguns simuladores utilizados para facilitar a compreensão e fornecer uma visão mais prática sobre os conceitos de que tratam são:

• *EasyCPU* - realiza simulação de arquiteturas de processadores [YYP01];

• *JFLAP* - simula autômatos finitos e a máquina de Turing [RBFR06];

• *Deus Ex Machina* - simula sete diferentes modelos de computação, entre eles, autômatos e a máquina de Turing [Sav06].

3.2 A importância da simulação para a computação quântica

Se a simulação oferece vantagens para áreas onde os sistemas simulados são concretos, ela é muito mais importante para áreas onde ainda não foi possível implementar o sistema desejado. Desse modo, para a computação quântica, a simulação se tornou uma das alternativas mais viáveis para o estudo e o desenvolvimento da área.

Atualmente os trabalhos relacionados ao desenvolvimento de simuladores têm produzido ferramentas, tais como simuladores de circuitos quânticos, simuladores de máquinas de Turing quânticas e linguagens de programação, os quais facilitam a compreensão de algum aspecto relacionado à computação quântica.

As linguagens de programação permitem descrever um algoritmo em um nível de abstração familiar à comunidade de engenharia de *software* e áreas afins. Por sua vez, um simulador de circuitos quânticos permite descrever (textualmente e/ou graficamente) um algoritmo em termos de portas e circuitos e testar esse algoritmo para um estado quântico específico através da simulação do *hardware* assim descrito. A linguagem de circuitos quânticos tem descrito os principais algoritmos quânticos conhecidos, ela é mais próxima dos físicos e dos engenheiros eletricistas, pois possui bastante similaridade com o seu análogo clássico que é amplamente conhecido.

Como a computação quântica é interdisciplinar, ou seja, envolve conhecimentos de física, matemática, computação, é salutar fornecer ferramentas que descrevam este paradigma de forma interessante para todas estas áreas.

Deve-se salientar que qualquer abordagem de simulação do paradigma computacional quântico em sistemas clássicos irá sofrer a mesma limitação: a ineficiência clássica para simular sistemas quânticos [Fey82]. Ainda assim, a disponibilidade de um sistema computacional que permita uma descrição em nível apropriado de um algoritmo quântico e uma "máquina" para executar (ou simular) o algoritmo, facilitam tanto o ensino quanto o próprio desenvolvimento de algoritmos.

Ferramentas de simulação são importantes devido a uma série de motivos, como os apresentados por [Wal99] e [CL03], entre os quais pode-se citar:

- a possibilidade de observação dos estados intermediários de uma computação, o que

em um computador quântico real seria indesejável, pois uma medição levaria ao colapso dos estados do sistema em estados da base de medição;

• a visualização de circuitos quânticos ou das instruções de códigos de uma linguagem de programação quântica, facilita sobremaneira a compreensão, e também estimula o surgimento de novos algoritmos;

• a fácil repetição e distribuição de experimentos, uma vez que alguns simuladores permitem que a descrição de uma simulação seja armazenada em arquivos para posterior execução.

Para o caso específico de simuladores de circuitos quânticos, as ferramentas de simulação são úteis em diversas situações tanto de ensino e aprendizagem quanto de desenvolvimento de algoritmos quânticos. Estas ferramentas permitem que:

• resultados sejam obtidos de maneira muito mais rápida do que se fossem realizados a mão;

• alunos e professores possam explorar a velocidade e testar diversas soluções possíveis em menos tempo;

• alunos identifiquem ações inválidas, pois as restrições de construção e edição dos circuitos não permitem a execução destas ações;

• pesquisadores possam investigar diversas alternativas para resolução do problema estudado;

• o aluno possa ter mais autonomia, pois a ferramenta pode ser utilizada em alguns momentos quando o professor não está disponível, por exemplo, na checagem dos resultados obtidos durante a resolução de exercícios;

• o aluno possa identificar os erros cometidos e buscar mais rapidamente o "caminho" correto para encontrar a solução desejada.

Os simuladores quânticos se tornam ainda mais importantes pois, com relação ao desenvolvimento de algoritmos e comparado ao caso clássico, possuem um grau de dificuldade maior, uma vez que o entendimento da computação quântica atualmente passa pela compreensão tanto dos aspectos físicos quanto do modelo matemático envolvidos na sua descrição. Existe ainda a dificuldade "cultural", como se pode observar na afirmação de Nielsen e Chuang [NC00] "Para projetar bons algoritmos quânticos, nós devemos "desligar" nossa intuição clássica, pelo menos parcialmente, e usar efeitos verdadeiramente quânticos para

chegarmos à proposta do algoritmo".

3.3 Simuladores simbólicos de circuitos quânticos

Entre os simuladores de circuitos existentes, as ferramentas pertencentes à categoria de simuladores simbólicos são as que melhor descrevem o estado do sistema simulado. Os simuladores simbólicos são, em sua totalidade, pacotes de extensão para *CAS*. Estas ferramentas utilizam as funcionalidades existentes nos sistemas algébricos para exibir representações do estado do sistema. O *QDensity* e o *QuCalc*, ambos para o *Mathematica* e o *OpenQUACS*, para *Maple*, são exemplos de simuladores simbólicos.

3.3.1 Simulador *QuCalc*

O *QuCalc* foi criado por Paul Dumais nos laboratórios de informática teórica e quântica na Universidade de Montreal e informática quântica da Universidade McGill, este pacote permite realizar simulações relacionadas à computação quântica, mas para isso é necessário conhecer o funcionamento do *Mathematica*.

Em geral, os pacotes de extensão vêm acompanhados de "tutoriais", os quais contém exemplos de uso das funcionalidades oferecidas. A Figura 3.1 exibe o tutorial no lado esquerdo e um exemplo de execução ao lado direito.

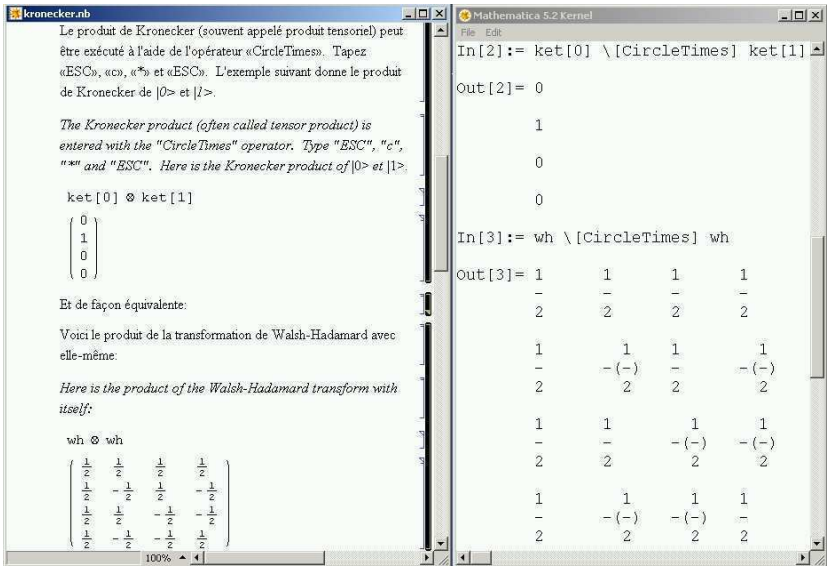


Figura 3.1: Exemplo de simulação no *Mathematica* utilizando o *QuCalc*.

No tutorial exibido na figura pode-se ver algumas explicações em inglês e francês sobre o produto de *Kronecker*, além de ilustrações de utilização. Na execução dos exemplos apresentados, tem-se o produto $|0\rangle \otimes |1\rangle$, representado por $ket[0]\backslash[CircleTimes]ket[1]$, além da aplicação do mesmo produto sobre o operador *Walsh-Hadamard*, representado por *wh*. Com estes exemplos pode-se afirmar que, para usar o pacote de maneira satisfatória, deve-se conhecer bem os comandos suportados, pois nenhum usuário deve intuitivamente conhecer a sintaxe utilizada para todos os conceitos existentes.

A abordagem utilizada pelo *QuCalc* consiste em definir elementos da linguagem de circuitos utilizando aqueles já presentes no *Mathematica*. Por exemplo, a definição de estados e operadores é realizada em termos de vetores e matrizes. Portanto os *kets* são representados como vetores coluna, enquanto transformações unitárias são matrizes quadradas. Isto posto, utilizando as definições fornecidas no *QuCalc* o usuário pode simular qualquer circuito quântico e observar, exclusivamente, a descrição simbólica do estado do sistema.

3.3.2 Simulador *QDensity*

O *QDensity* foi criado por Bruno Juliá-Diaz, Joseph M. Burdis e Frank Tabakin, nas Universidades do Estado da Carolina do Norte e de Pittsburgh. Assim como no *QuCalc*, a definição dos conceitos suportados pelo *QDensity* se baseia naqueles presentes no *Mathematica*, com isso é possível representar os estados quânticos através de vetores ou matriz de densidade, sendo a segunda forma o padrão de representação.

Nas simulações executadas no *QDensity* apenas a descrição simbólica do estado do sistema é fornecida, não sendo oferecida nenhuma forma de representação gráfica do circuito.

O pacote também possui tutoriais sobre alguns temas relacionados à computação quântica, entre estes, tem-se: Teleportação, Transformada de Fourier Quântica (*QFT*), Algoritmo de Grover e Algoritmo de Shor. A Figura 3.2 exibe um tutorial sobre o algoritmo de Shor no lado esquerdo, a tela ao lado direito exibe a execução do mesmo exemplo apresentado para o *QuCalc*, sobre o qual podem ser feitas as mesmas considerações.

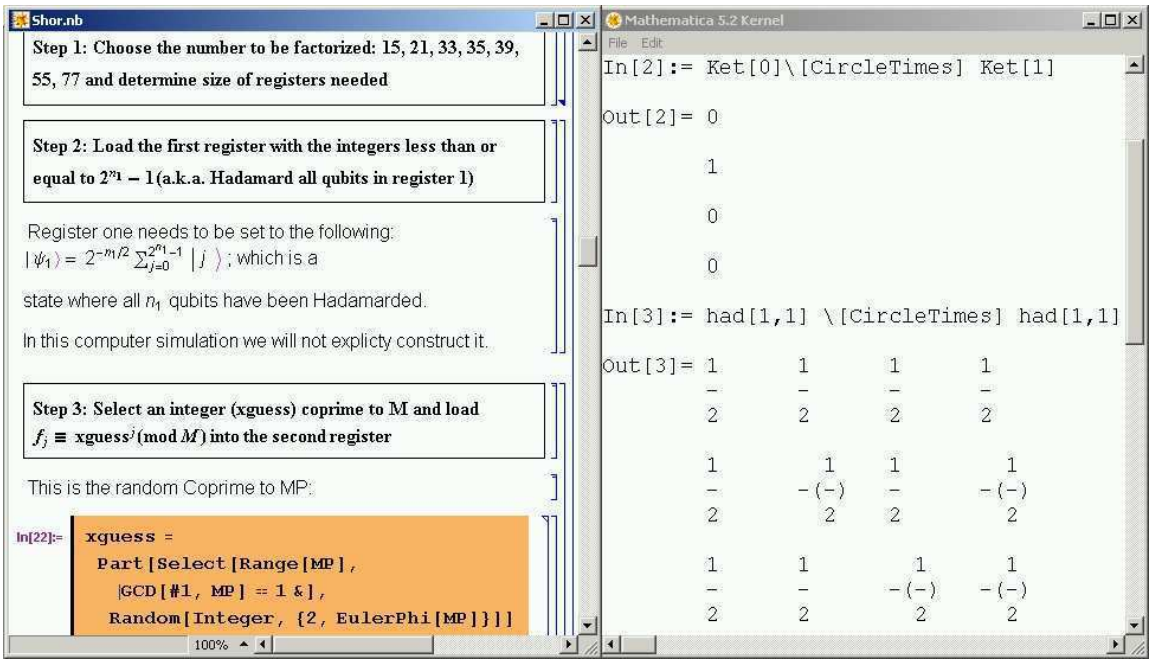


Figura 3.2: Exemplo de simulação no *Mathematica* utilizando o *QDensity*.

3.3.3 Simulador *OpenQUACS*

O *OpenQUACS* foi criado por Christopher B. McCubbin, Samuel J. Lomaco e Lallane Kagal, na Universidade do Condado de Maryland. Do mesmo modo que nos simuladores já citados, os conceitos da computação quântica existentes no pacote são definidos com base em outros já suportados pelo *Maple*. Para que o usuário se familiarize com os conceitos fornecidos, o *OpenQUACS* é distribuído juntamente com um tutorial que é dividido em treze lições, nas quais são apresentadas as funcionalidades existentes no simulador.

A Figura 3.3 exibe parcialmente a janela do *Maple* onde são executados alguns comandos disponíveis no simulador, tais como:

- *Make_Ket[valor, dimensão, base]* - cria um qubit com o valor e dimensão especificados para a base informada, exibindo o mesmo em notação matricial;
- *RKet[qubits]* - exibe os qubits utilizando a notação de Dirac;
- *Hadamard(n)* - cria o produto tensorial de “n” operadores Hadamard.

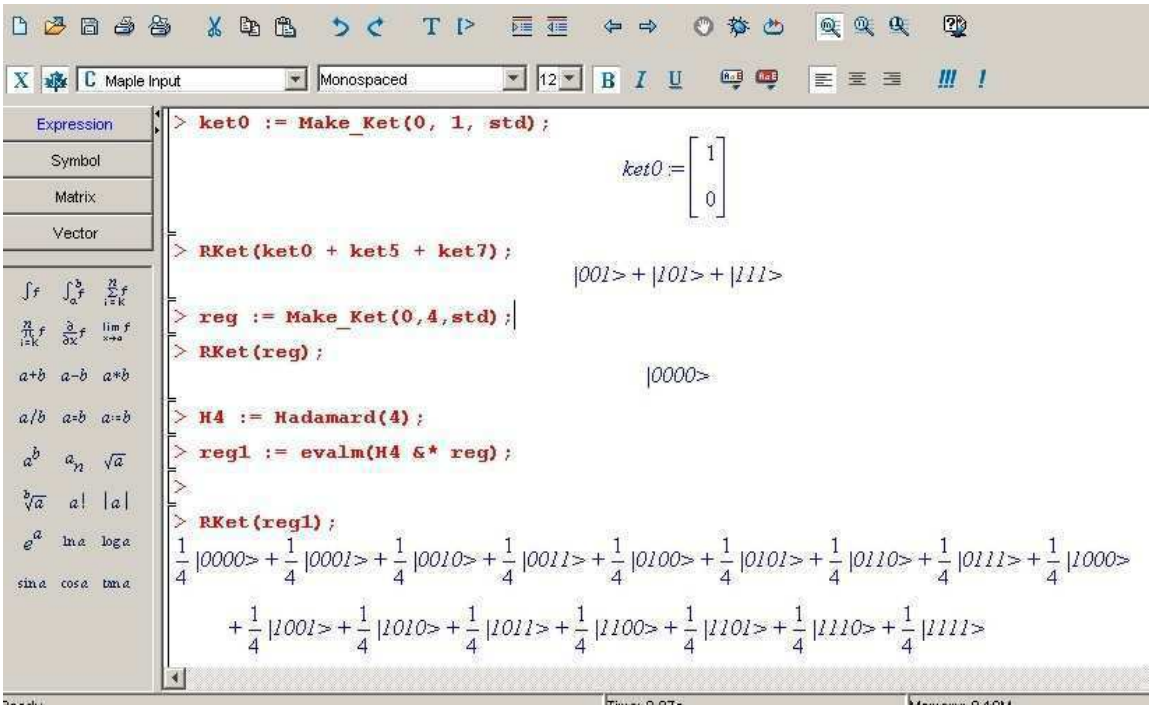


Figura 3.3: Exemplo de simulação no *Maple* utilizando o *OpenQUACS*.

3.4 Simuladores universais de circuitos quânticos

Os simuladores universais fornecem a representação gráfica do circuito e uma descrição numérica do estado do sistema. Algumas ferramentas desta categoria possibilitam manipulação direta dos componentes, nestas ferramentas um circuito quântico é descrito por uma representação visual, que inclui suas portas e conexões, já o estado do sistema possui uma descrição numérica que exprime sua evolução ao longo do circuito.

Os simuladores *jaQuzzi* [Sch00], *QuaSi* [Mat06], *Senko Quantum Simulator* [Sen06] e *Zeno* [CLL05] são exemplos de simuladores universais.

3.4.1 Simulador *jaQuzzi*

O *jaQuzzi* é um simulador gratuito desenvolvido em Java, portanto multiplataforma, que foi implementado na dissertação de mestrado de Felix Schürmann pela Universidade do estado de Nova York em 2000. Esta ferramenta busca fornecer à comunidade acadêmica um meio para facilitar o projeto e a visualização de circuitos quânticos, oferecendo a descrição numérica do estado do sistema aliada à visualização do circuito implementado. Entretanto,

apesar de possuir interface gráfica, a edição de um circuito não pode ser realizada através de manipulação direta dos componentes, o que torna necessário a utilização de menus e formulários.

A Figura 3.4 exibe um exemplo de simulação no *jaQuzzi*, na tela superior é possível ver a janela de edição de circuitos e na tela inferior a janela de visualização do estado do sistema. Uma observação que pode ser realizada se refere ao símbolo de medição utilizado no simulador, que não corresponde ao símbolo padrão usado na linguagem de circuitos.

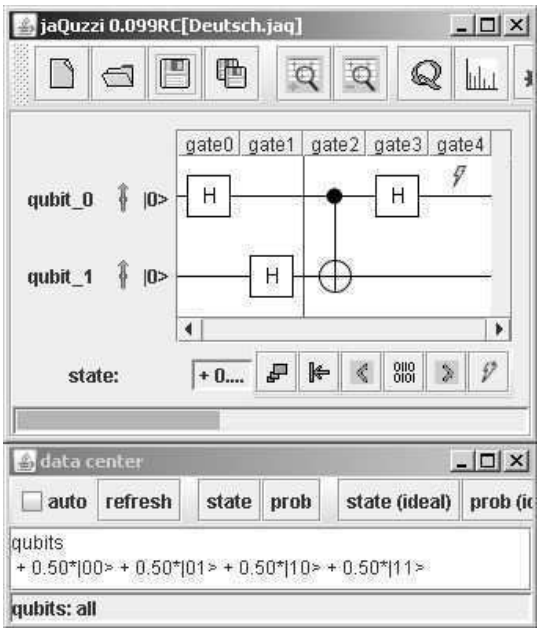


Figura 3.4: Exemplo de simulação no simulador universal *jaQuzzi*.

Essa ferramenta pode ser executada como uma aplicação ou como um *applet* (um programa executado em um navegador web). Algumas das características do *jaQuzzi* são: a possibilidade de simulação de erros internos (operacionais) e externos (descoerência); e os modos de execução (interativo ou em lotes).

3.4.2 Simulador *QuaSi*

O *QuaSi* (*Quantum Simulator*) foi desenvolvido na Alemanha no Instituto de Algoritmos e Sistemas Cognitivos da Universidade de Karlsruhe. Assim como o *jaQuzzi*, este simulador pode ser executado como uma aplicação ou como um *applet* e, por ser implementado na linguagem Java, também é multiplataforma.

A Figura 3.5 exibe as telas de projeto de circuitos à esquerda e de visualização do estado do sistema à direita. O *QuaSi* também é um simulador exclusivamente numérico, como pode ser visto na descrição do estado do sistema. Algumas críticas podem ser observadas: os estados iniciais são apresentados unicamente através de rótulos, o que dificulta a identificação do estado dos qubits; a representação gráfica de uma medição também não é a representação padrão apresentada na literatura.

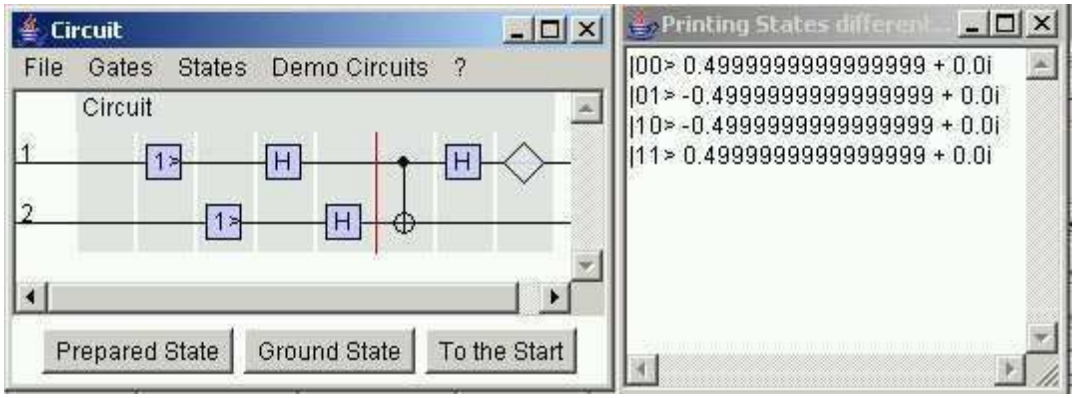


Figura 3.5: Exemplo de simulação no simulador universal *QuaSi*.

O simulador *QuaSi* possui interface gráfica, porém como acontece no *jaQuzzi*, ele não permite a manipulação direta de componentes, sendo assim é necessário utilizar menus e formulários para projetar um circuito. Além disso, o *QuaSi* não permite desfazer ações ou remover portas e qubits. O fato de não permitir desfazer ações e manipular os componentes de circuito diretamente torna trabalhoso o processo de implementação de algoritmos.

3.4.3 Simulador *Senko's Quantum Computer*

O *Senko's Quantum Computer* desenvolvido no Japão foi o primeiro simulador universal implementado. Este simulador é uma ferramenta comercial que possibilita a construção de circuitos através de uma interface gráfica e permite a edição do circuito através de manipulação direta dos seus elementos. Assim como os outros simuladores descritos, o *Senko's Quantum Computer* fornece a representação gráfica do circuito aliada à descrição numérica do estado do sistema. Na Figura 3.6 vê-se um exemplo de simulação, na tela superior é exibida a representação gráfica do circuito, na inferior vê-se a descrição numérica do estado do sistema.

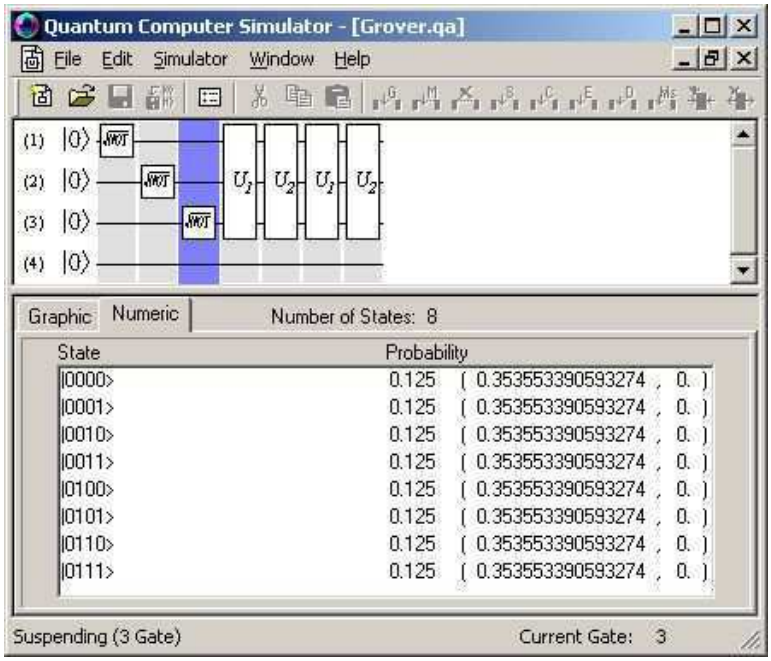


Figura 3.6: Exemplo de simulação no simulador universal *Senko's Quantum Computer*.

Além da representação numérica, o simulador permite também que o estado do sistema seja representado simbolicamente, porém somente se este for utilizado em conjunto com o *Mathematica*. Para realizar a simulação simbólica, o usuário deve exportar o circuito desejado como um *Mathematica notebook*, com isso o usuário pode visualizar no *CAS* a descrição simbólica do estado do sistema.

3.5 Deficiências

Apesar de serem ferramentas úteis, todos os simuladores apresentam deficiências no que diz respeito à sua utilização tanto em contextos de ensino e aprendizagem quanto no próprio desenvolvimento de algoritmos quânticos.

Os simuladores simbólicos são unicamente simbólicos, ou seja, não oferecem nenhum tipo de representação gráfica de circuitos. As ferramentas desta categoria, que são em geral gratuitas, possibilitam a descrição e simulação de circuitos quânticos, porém só podem ser executadas no *CAS* onde foram criadas, que em geral não é gratuito. Com isso é necessário conhecer o funcionamento do *CAS* para só então utilizar esses pacotes de maneira satisfatória.

Os simuladores universais, por sua vez, apesar de possibilitarem a descrição gráfica do circuito, oferecem apenas uma representação numérica do estado do sistema. Esta forma de representação não se caracteriza como a forma mais adequada, visto que é muito mais natural efetuar os cálculos relacionados de maneira simbólica, tal como é apresentado na literatura.

O *Senko's Quantum Computer* realiza simulação simbólica, mas o faz somente em conjunto com o *Mathematica*, ou seja, na verdade o *Senko's Quantum Computer* apenas delega o processamento simbólico. Com isso, essa alternativa leva as mesmas desvantagens citadas anteriormente. Como simulador universal, o *Senko's Quantum Computer* é exclusivamente numérico e como simulador simbólico se faz necessária a aquisição e aprendizagem do *Mathematica*.

Como mostra a literatura sobre computação quântica, o entendimento de um algoritmo quântico na linguagem dos circuitos passa pela representação gráfica do circuito (sintaxe) aliada à descrição e à manipulação simbólica do estado do sistema (semântica). A representação matemática do estado do sistema e sua transformação, quer pela aplicação de uma operação (unitária ou de projeção), quer pela sua simplificação ou expansão algébrica, são elementos fundamentais tanto na compreensão do problema quanto na obtenção de sua solução através de um algoritmo quântico. Assim, pode-se afirmar que as ferramentas descritas não são as mais adequadas para simulação da linguagem de circuitos quânticos.

Capítulo 4

O simulador Zeno

Neste capítulo, as funcionalidades originais do simulador de circuitos quânticos Zeno são ilustradas, ou seja, o simulador é apresentado sem a extensão desenvolvida neste trabalho. Além disso, são descritas também a organização interna do simulador, as modificações, atualizações e correções anteriores ao desenvolvimento da extensão.

4.1 Versão original do simulador

O simulador *Zeno* original foi desenvolvido na UFCG como dissertação de mestrado de Gustavo E. M. Cabral. O *Zeno* é um simulador universal de circuitos quânticos que permite o projeto e a edição de algoritmos. A Figura 4.1 exibe a interface gráfica da versão original do simulador, formada pelos seguintes componentes:

1. Editor de circuitos;

2. Janela de observação do estado do sistema.

A usabilidade é um dos principais diferenciais desta ferramenta, a interação do usuário com o simulador pode ocorrer através de menus, formulários e botões disponíveis na interface. É possível também projetar um circuito através da manipulação direta de componentes, característica inexistente em muitos simuladores. A manipulação direta dos componentes de um circuito facilita sobremaneira a implementação de algoritmos.

Uma das principais vantagens associadas à utilização de simuladores de circuitos quânticos é a possibilidade de observação de estados intermediários de uma computação. A janela

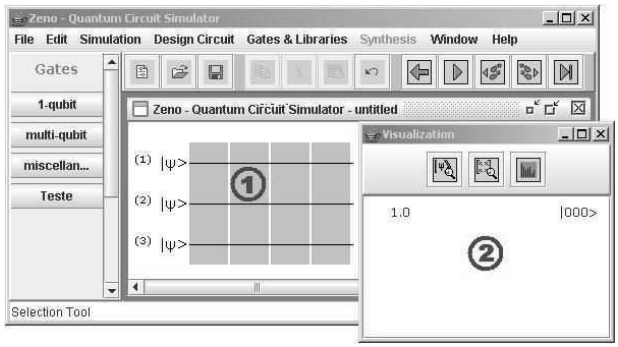
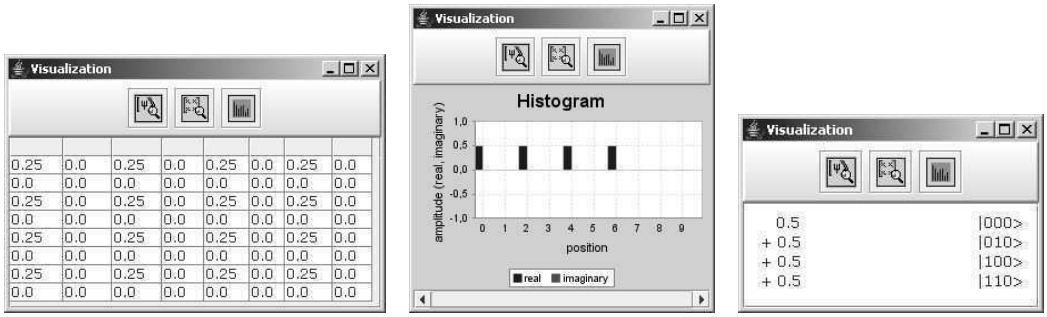


Figura 4.1: *GUI* original do simulador *Zeno*.

de observação do estado permite três opções de visualização, sendo elas: a representação matricial, a visualização através de histograma e a representação numérica utilizando a notação de Dirac (ou *braket*). A Figura 4.2 ilustra cada uma das opções de visualização.



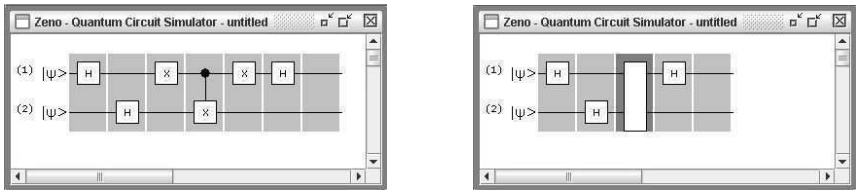
(a) Notação matricial (b) Histograma (c) Notação de Dirac

Figura 4.2: Opções de visualização do estado do sistema no *Zeno*

O *Zeno* permite que sejam escolhidos estados puros ou mistos para realização de uma simulação, em outras ferramentas apenas uma destas opções é oferecida. Após a definição sobre qual tipo de estado será usado, um algoritmo pode ser implementado utilizando o conjunto pré-definido de portas ou através da criação de operadores que facilitem a construção do circuito. Os operadores podem ser organizados em bibliotecas personalizáveis de acordo com a necessidade do usuário, os tipos de portas suportados pela ferramenta são: matricial, rotação, medição, traço e *ancilla*.

Muitas vezes a implementação de um algoritmo emprega o uso de oráculos para representar uma função qualquer, na literatura o uso destes elementos é bastante comum. Uma funcionalidade interessante presente no *Zeno* é o agrupamento de colunas (e das portas exis-

tentes nas colunas), que possibilita a representação de oráculos e o uso de uma menor área de visualização. A Figura 4.3 exibe um circuito que emprega o uso de um oráculo.



(a) Circuito sem colunas agrupadas (b) Circuito com colunas agrupadas

Figura 4.3: Agrupamento de colunas no *Zeno*

O *Zeno* oferece ainda ações de copiar, recortar, colar e desfazer, funcionalidades básicas existentes em inúmeras aplicações, mas que não são fornecidas por vários simuladores. A ausência de tais ações dificulta a criação e edição de circuitos, desse modo, em algumas ferramentas cometer um erro na criação de um circuito significa voltar ao início de sua implementação.

Logo, pode-se inferir que o *Zeno* possui características que englobam o conjunto de funcionalidades existente em outras ferramentas do gênero, sendo assim de grande valor para estudantes e pesquisadores da área de computação e informação quântica. Contudo, assim como os outros simuladores, o *Zeno* em sua versão original era um simulador exclusivamente numérico.

4.2 Modelos e diagramas

Internamente a versão original do simulador *Zeno* não sofreu nenhuma modificação relacionada a seu projeto, em alguns casos apenas mudanças na forma de execução de funcionalidades foram realizadas. Nesta seção, são apresentados os principais modelos e diagramas exibidos em [Cab04] relacionados à versão original do simulador.

4.2.1 Arquitetura

O modelo de arquitetura de uma sistema exibe os principais "blocos" existentes no mesmo. A arquitetura do simulador *Zeno* é representada na Figura 4.4, onde é possível observar a in-

dependência entre as camadas de interface e aplicação. A ferramenta possibilita que circuitos e bibliotecas sejam armazenados para uso posterior ou para distribuição. Além disso, vê-se também que o simulador utiliza um engenho matemático, empregado devido à necessidade de representação de estruturas, em geral, inexistentes nas linguagens de programação, como por exemplo os números complexos e as matrizes.

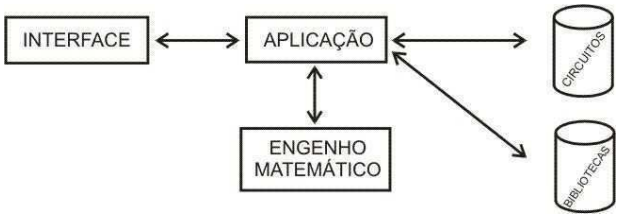


Figura 4.4: Modelo arquitetural do simulador *Zeno*

4.2.2 Modelo conceitual

Um modelo conceitual exibe as principais entidades do sistema, mostrando também o tipo de relacionamento existente entre elas. No simulador *Zeno* as principais entidades são aquelas relacionadas à representação de estados e portas quânticas, bem como à representação de circuitos. O modelo conceitual do sistema pode ser observado na Figura 4.5.

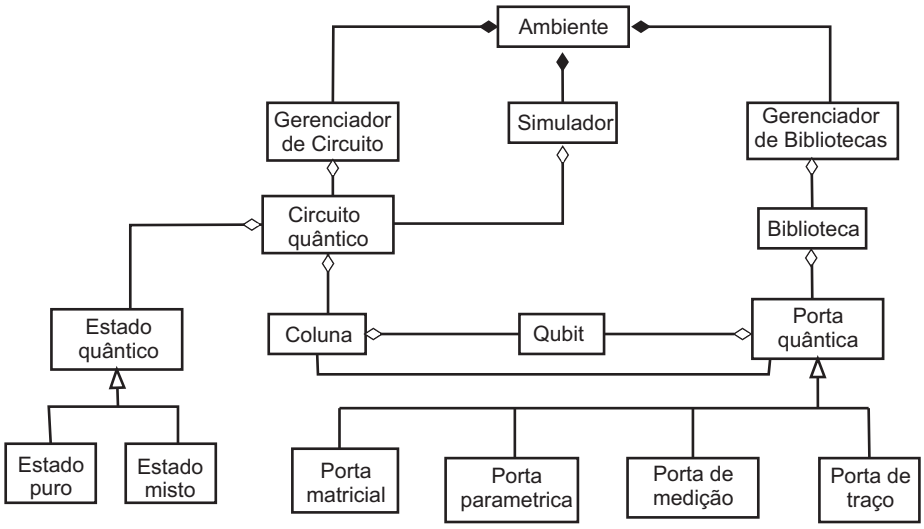


Figura 4.5: Modelo conceitual do simulador *Zeno*

4.2.3 Diagrama de classes

Diagramas de classe detalham as entidades envolvidas na implementação de um sistema, correspondendo assim a uma representação estática da estrutura do sistema. Estes diagramas exibem as classes e suas relações, bem como as ações e atributos pertencentes a cada uma das classes representadas. Devido às suas dimensões, os diagramas de classe do *Zeno* não são adequados para exibição neste trabalho, estes diagramas podem ser visualizados em [IQU07].

4.3 Refatoramento

Foram realizadas algumas modificações para facilitar o desenvolvimento do *CAS* e a união deste com o editor de circuitos do *Zeno*. Uma descrição detalhada de cada uma destas alterações é apresentada em [BLL06], apenas um resumo das mudanças efetuadas é exibido nesta seção. É importante salientar que nenhuma nova funcionalidade foi criada durante as alterações, apenas atividades de refatoramento e correção de erros foram realizadas. Também são apresentadas algumas das mudanças relacionadas à execução de funcionalidades, realizadas para facilitar o desenvolvimento da extensão.

As modificações feitas sobre a versão original do simulador podem ser classificadas em dois grupos: aquelas relacionadas à organização do sistema e outras relacionadas à atualização deste. As do primeiro grupo tiveram como objetivo facilitar a compreensão do simulador como um todo, bem como a leitura de seu código, simplificando assim o entendimento de funcionalidades específicas. As atualizações tiveram como objetivo melhorar diversos aspectos do *Zeno*, como por exemplo a substituição dos componentes utilizados por versões mais atuais.

Foram incluídas também algumas ferramentas de apoio ao desenvolvimento, dentre elas tem-se: *Cobertura* [Dol06], *Checkstyle* [Bur05], *Checkeclipse* [Mee05] e *ANT* [Apa06], as quais podem ajudar na realização de atividades de validação do sistema, facilitar a execução de tarefas ao longo do ciclo de desenvolvimento ou ainda ajudar o(s) desenvolvedor(es) na adequação do código às convenções estabelecidas.

As organizações de código não causariam adição de erros ou qualquer modificação nas funcionalidades existentes. Portanto, não foi necessário realizar nenhuma etapa de valida-

ção para garantir que as modificações não fossem prejudiciais ao simulador. As alterações classificadas como organização do código são:

- adoção de convenções de código;
- exclusão de trechos de código;
- modificação da estrutura de pacotes e pastas do *Zeno*;
- adoção de ferramentas de apoio ao desenvolvimento.

Por sua vez, algumas das atualizações poderiam causar a adição de erros ou modificar funcionalidades existentes. Por isso, foi necessário realizar a validação destas alterações para garantir que nenhuma delas prejudicasse o simulador. As modificações classificadas como atualizações são:

- adequação do código fonte as novas características da *J2SE 5.0* (*Tiger*) [Sun06b];
- substituição de métodos desaprovados;
- análise e substituição de versões antigas de componentes por versões mais atuais;

Antes de realizar qualquer uma das modificações citadas, foram analisados os testes unitários originais contidos na aplicação, após a análise foram criados testes de cobertura. Novos casos de teste foram implementados com o intuito de aumentar a quantidade de código validada pelos casos de teste.

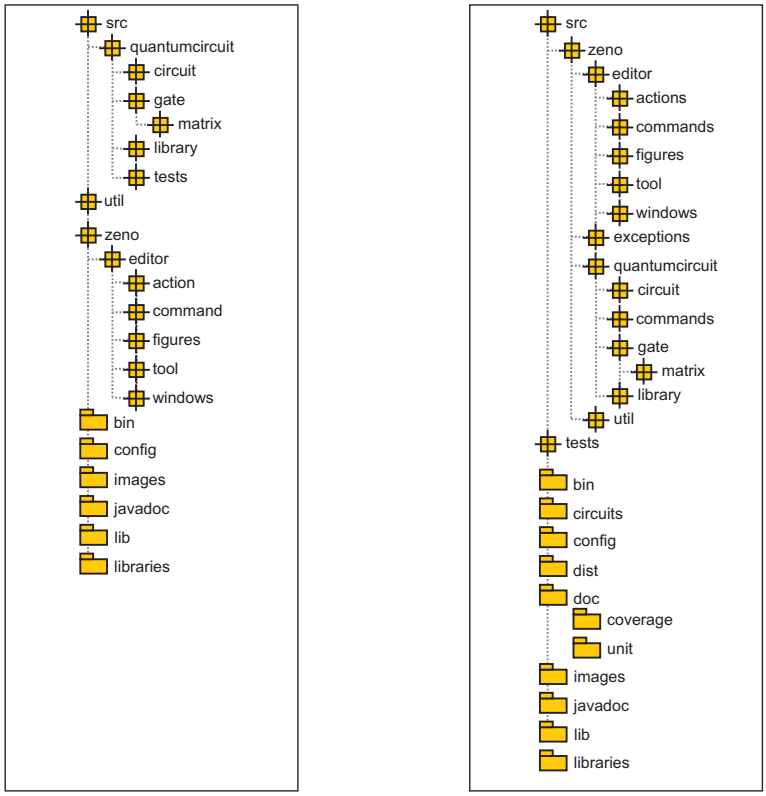
4.3.1 Reorganizações

As convenções de codificação da *Sun Microsystems*, denominadas de *Sun Code Conventions* [Sun06a] para a linguagem Java, foram parcialmente adotadas. Padrões de codificação são importantes, pois melhoram a leitura do código e facilitam sua compreensão.

A exclusão de trechos de código do simulador teve como objetivo diminuir o tamanho total de cada arquivo, facilitando a leitura e como conseqüência a compreensão das funcionalidades implementadas. Foram excluídos diversos trechos de código que se encontravam comentados, bem como outros códigos irrelevantes para o correto funcionamento da ferramenta. Algumas classes que não eram utilizadas em nenhum aspecto do simulador também foram excluídas.

A estrutura de pacotes e pastas originalmente adotada no simulador foi alterada, tal como exibido na Figura 4.6, com isso alguns problemas e desvantagens foram eliminados. A nova organização estrutural facilita o entendimento e o desenvolvimento de funcionalidades, uma

vez que agrupa as classes de acordo com suas características.



(a) Estrutura original de pacotes (b) Estrutura modificada de pacotes

Figura 4.6: Estrutura de pacotes do simulador *Zeno*

Inúmeras ferramentas de apoio ao desenvolvimento podem ser utilizadas em conjunto com a linguagem Java. Algumas das ferramentas adotadas no *Zeno* são:

• *JUnit* [Obj04] - é um arcabouço para realização de testes unitários automáticos;

• *Cobertura* [Dol06] - identifica o percentual de código testado na aplicação, além de exibir os trechos que foram e os que não foram testados;

• *Checkeclipse* [Mee05] e *Checkstyle* [Bur05] - indicam se o código fonte obedece ao padrão de codificação adotado;

• *ANT* [Apa06] - automatiza a execução de inúmeras atividades relacionadas ao desenvolvimento, como por exemplo a compilação do código fonte e a execução de testes.

4.3.2 Atualizações

Antes de realizar qualquer atualização, os testes originais contidos na aplicação foram analisados, após esse procedimento foram feitas as correções necessárias e só então as atualizações foram executadas.

O código original do simulador *Zeno* foi desenvolvido para a *J2SE* 1.4, durante o trabalho foi lançada a versão *J2SE* 5 (ou 1.5) e hoje a versão mais atual é a 6 (ou 1.6). Com o intuito de utilizar as vantagens da *J2SE* 5, foram feitas diversas modificações sobre o código do simulador. Além disso, foram substituídos elementos desaprovados, ou seja, elementos ainda suportados mas de uso desaconselhado.

O simulador *Zeno* adota alguns componentes na execução de suas funcionalidades, desde sua criação algumas versões mais novas de tais componentes foram lançadas. As atualizações foram executadas devido a uma série de motivos, como por exemplo correção de erros ou melhoria de desempenho. Os componentes utilizados pelo *Zeno* são:

• *JFreeChart* [GM06] - empregado na geração de gráficos para a visualização de histograma. A versão 0.9.19 presente na ferramenta original, foi substituída pela versão 1.0.1;

• *JHotDraw* [GQK04] - arcabouço para construção de aplicações gráficas, é usado na criação de toda a interface gráfica. A versão estável (5.3) mais recente disponível na data das refatorações é utilizada no simulador, outras versões eram disponibilizadas na época, porém por não serem estáveis elas não foram utilizadas;

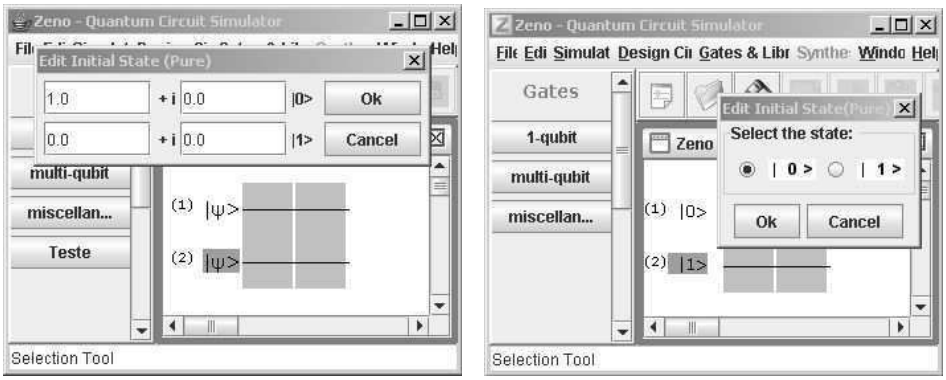
• *Java Addition to Default Environment* (*J.A.D.E*) [Dau06a] - engenho matemático responsável pela representação de operadores e estados quânticos. O projeto original do componente foi dividido em dois outros *JScience* [Dau06c] e *Javolution* [Dau06b]. A estrutura destes projetos foi bastante alterada, sendo assim foi necessário modificar o código do simulador. Para solucionar os problemas relacionados a não compatibilidade entre *J.A.D.E.* e *JScience* foi criada a classe utilitária *zeno.util.MatrixOperations*, a qual implementa os métodos não contidos ou alterados no *JScience*.

Após a realização de cada uma das atualizações, os testes implementados foram executados e quando houve algum caso de teste falho foram realizadas as correções necessárias.

Mudanças na implementação de funcionalidades

Na versão original do simulador a criação e a edição de estados e portas quânticas podia receber como valores de entrada números reais. Como não existe um processo para conversão de um valor real qualquer para seu equivalente simbólico (e.g. $0,3333 \rightarrow \frac{1}{3}$ ou $1,414221 \rightarrow \sqrt{2}$), foi necessário alterar a implementação de tais funcionalidades.

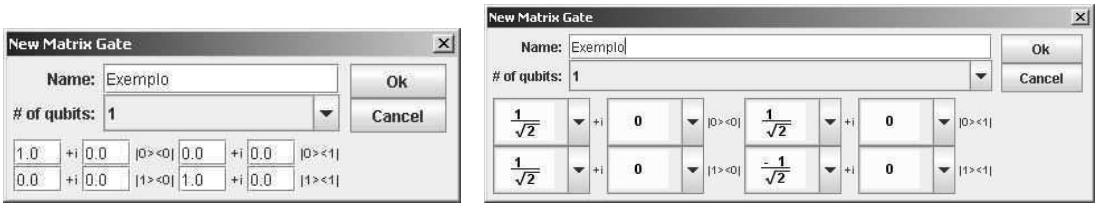
Após a modificação a edição de estados iniciais passou a aceitar apenas estados da base canônica como entrada, os rótulos $|\psi\rangle$, que representavam um estado de entrada qualquer, foram substituídos pela representação literal do estado. A Figura 4.7 ilustra a representação e a edição do estado inicial em uma simulação.



(a) Original (b) Modificado

Figura 4.7: Editar estado inicial

Outro exemplo de alteração na forma de execução de funcionalidades se refere à criação e edição de portas. Atualmente a definição de operadores aceita apenas valores pré-definidos como entrada. A Figura 4.8 ilustra a criação e a edição de uma porta matricial.



(a) Original (b) Modificado

Figura 4.8: Criar e editar portas

Capítulo 5

A extensão do simulador Zeno

Neste capítulo, é exibida a extensão desenvolvida ao longo deste trabalho. Serão apresentados os dois principais componentes do *CAS* do simulador, o mecanismo de representação associado à interface com o usuário e o mecanismo de manipulação de expressões. As funcionalidades suportadas serão também ilustradas através de exemplos de utilização.

5.1 A interface gráfica com o usuário e o mecanismo de representação de expressões

A extensão do *Zeno* não alterou a forma de execução de uma simulação, esta ação pode ser realizada da mesma maneira que no simulador original. Isso ocorre pois o *CAS* pode ser "ligado" e "desligado" de acordo com a necessidade do usuário, após a inicialização do sistema as manipulações são executadas somente após a solicitação do usuário. Com isso, garante-se que o desempenho do simulador não foi alterado.

A tela do *CAS*, mostrada na Figura 5.1, contém as seguintes áreas:

1. Barra de menus - possui todos os menus do sistema, através dos quais é possível alterar diversas propriedades de exibição, executar simulações e manipulações;

2. Barra de ferramentas - possui botões que facilitam a execução de ações mais freqüentes. Diversos botões existentes no editor de circuitos estão disponíveis na barra de ferramentas do *CAS*. Com isso, o usuário pode realizar diversas ações sem que seja necessário navegar do *CAS* para o editor ou vice-versa. Também estão presentes nesta

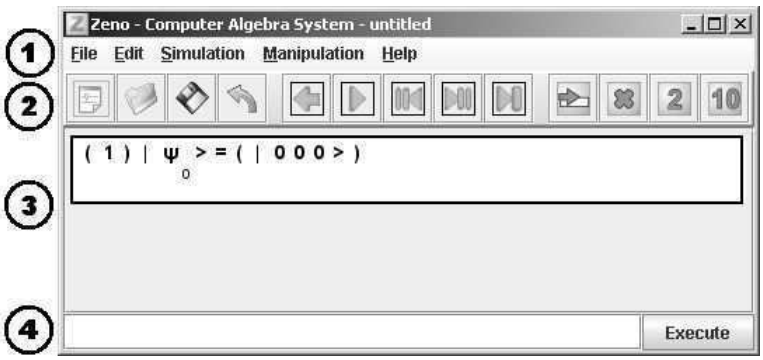


Figura 5.1: Tela do *CAS* do *Zeno*

barra os botões de comandos do sistema, por meio dos quais se pode solicitar a execução das diversas manipulações suportadas. É possível selecionar quais comandos estarão disponíveis na barra;

3. Área de expressões - contém todas as expressões relacionadas a uma simulação. Após cada passo de simulação ou execução de uma manipulação uma nova expressão é adicionada à área de expressões;

4. Prompt de comandos - executa todos os comandos suportados pelo simulador. Exibe os comandos que podem ser executados à medida que o usuário digita a sintaxe dos mesmos.

A exibição de cada uma das expressões presentes no sistema ocorre através da utilização do mecanismo de representação de expressões. Este mecanismo foi totalmente implementado ao longo deste trabalho, pois nenhuma das ferramentas estudadas se mostrou adequada aos requisitos do *CAS*. Os principais requisitos necessários para que uma ferramenta de representação seja adequada ao sistema são os seguintes:

- É imperativo que seja livre (*freeware* ou de código aberto *open-source*);
- É imperativo que seu registro seja compatível com a *GNU General Public License - GPL* [Fre07a];
- É necessário que seja implementada em Java ou possa ser utilizada a partir de uma aplicação Java;

- É desejável que se possa configurar os atributos de exibição, ou seja, deve-se poder alterar tamanho, estilo e cor da fonte, além de outras propriedades;
- É preferível que o código fonte possa ser modificado, pois alguma funcionalidade necessária pode não estar presente;
- É desejável que a sintaxe utilizada para criação de expressões seja amplamente conhecida;
- É necessário que exista seleção de partes de uma expressão.

5.1.1 Mecanismos de representação de expressões

Diversas ferramentas para representação de expressões foram estudadas. A grande maioria das ferramentas avaliadas oferecem soluções para exibir expressões matemáticas em páginas da internet, o que não é a abordagem ideal para o *CAS*. A principal restrição ao uso de uma ferramenta de representação de expressões é a compatibilidade com a licença *GPL*. Por tanto, nenhuma ferramenta comercial será descrita, apesar de algumas terem sido avaliadas.

ASCIIMathML

Trata-se de uma aplicação *javascript* que fornece meios para exibir expressões matemáticas em páginas da internet. Ferramentas que possuem esta abordagem não são as mais adequadas para o *CAS*. Todavia, estas foram avaliadas, pois, caso atendessem aos requisitos, seriam soluções válidas. Porém, seria necessário criar algum meio de comunicação entre a aplicação Java e o mecanismo de representação.

ASCIIMathML [Jip05] é uma ferramenta de código aberto, distribuída sob a licença *GPL*, e que pode ter seu código modificado. A sintaxe utilizada é uma formalização da linguagem de descrição matemática utilizada em texto puro. Logo, pode-se afirmar que em algum momento a padronização adotada na ferramenta não será de conhecimento público amplo.

O funcionamento da *ASCIIMathML* consiste em realizar uma tradução da palavra (*string*) de entrada, fornecida em sintaxe *ASCII*, para a linguagem *MathML* [Bos07], que pode ser interpretada por um navegador que suporte a mesma.

A Figura 5.2 mostra algumas expressões exibidas com a utilização do *ASCIIMathML*.

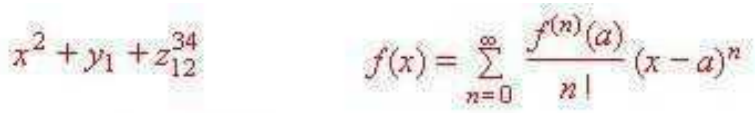


Figura 5.2: Exemplo de representação de expressões utilizando *ASCIIMathML*

HotEqn

HotEqn é um *applet* Java criado para exibir expressões matemáticas na internet. Por ser implementada na linguagem Java, o uso desta ferramenta a partir da aplicação seria facilitado. Contudo, apesar de ser uma aplicação livre, o código da mesma não pode ser alterado.

A sintaxe utilizada é um subconjunto da linguagem *LaTeX* [Wik07]. Este subconjunto inclui os elementos e notações utilizados comumente na matemática. Por utilizar a sintaxe *LaTeX*, pode-se afirmar que o uso da ferramenta é facilitado, uma vez que a linguagem é bastante conhecida.

O funcionamento da *HotEqn* consiste em traduzir a palavra de entrada, fornecida em sintaxe *LaTeX*, e gerar uma imagem de saída. As fontes disponíveis na aplicação correspondem a imagens de cada um de seus caracteres. Assim, cada *token* de entrada é transformado no símbolo correspondente utilizando determinada imagem. Visto isso, é possível afirmar que a modificação das configurações de exibição é bastante restrita, por exemplo, apenas os tamanhos de fonte 18, 14, 12, 10 e 8 podem ser utilizados.

A Figura 5.3 mostra algumas expressões exibidas com a utilização do *HotEqn*.

Figura 5.3: Exemplo de representação de expressões utilizando *HotEqn*

Swift

A *Swift* tem como objetivo facilitar a criação de expressões e a exibição destas em diversos contextos. A ferramenta possui dois componentes, são eles:

- Editor - é uma aplicação Java, desenvolvido em J2SE 1.1.3, utilizada para criar e editar expressões. A Figura 5.4 mostra o editor e a barra de ferramentas, onde são fornecidos

botões, que correspondem aos símbolos que podem ser utilizados.

- Visualizador - é um *applet* Java, desenvolvido em J2SE 1.0, que tem como função exibir as expressões criadas no editor. Trata-se de uma versão do editor onde não são permitidas alterações.

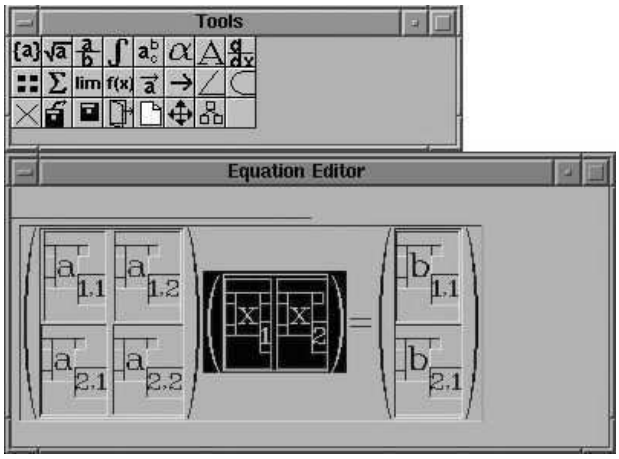


Figura 5.4: Exemplo de representação de expressões utilizando *Swift*

A ferramenta pode ser utilizada livremente, desde que não seja para fins comerciais. Seu código fonte não pode ser modificado, e não é possível alterar as configurações de exibição ou seleção de partes de uma expressão.

O funcionamento da ferramenta é similar ao da *HotEqn*. Uma diferença é que na *HotEqn* são utilizadas palavras de entrada em sintaxe *LaTeX*, enquanto na *Swift* as expressões são criadas no editor. Também são empregadas imagens para representar as fontes disponíveis, limitando as possibilidades de visualização.

ShowMath

A *ShowMath* é um *applet* Java criado para exibir expressões matemáticas em páginas da internet. Por ser implementada na linguagem Java, o uso desta ferramenta a partir do *Zeno* é facilitado. É uma aplicação de código livre, portanto, pode ter seu código alterado. Todavia, não é informada qual a licença utilizada na mesma, sendo assim, não se pode afirmar se há ou não compatibilidade com *GPL*.

A sintaxe utilizada é um subconjunto da linguagem *LaTeX*. Sendo assim, pode-se afirmar que o uso da ferramenta é facilitado, uma vez que a linguagem é bastante conhecida.

A Figura 5.5 mostra algumas expressões exibidas com a utilização do *ShowMath*.

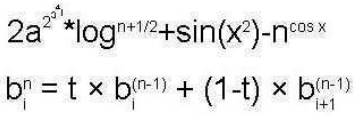


Figura 5.5: Exemplo de representação de expressões com *ShowMath*

O funcionamento da *ShowMath* consiste em traduzir uma palavra de entrada, em sintaxe *LaTeX*, para uma palavra composta de caracteres *unicode*. Esta solução é mais eficiente do que aquelas baseadas em imagens para exibição de caracteres. Ainda assim, existe a necessidade de possuir a fonte utilizada pela ferramenta para garantir a exibição correta das expressões.

Resumo comparativo das ferramentas

A Tabela 5.1 resume os requisitos suportados pelas ferramentas descritas anteriormente.

Tabela 5.1: Resumo comparativo das ferramentas de exibição de expressões

	ASCIIMath	*HotEqn*	*Swift*	*ShowMath*
Freeware ou *open-cource*	✓	✓	✓	✓
Compatível com *GPL*	✓			
Compatível com aplicação Java		✓	✓	✓
Configurável	✓	✓		
Pode ser modificada	✓			✓
Sintaxe é amplamente conhecida		✓		✓
Selecionável	✓	✓		

5.1.2 O mecanismo de representação adotado no *CAS*

Visto que nenhuma ferramenta existente possui todos os requisitos exigidos, fez-se necessário criar uma nova ferramenta. Como este tipo de aplicação é útil aos mais diversos sistemas, a implementação da nova ferramenta ocorreu de maneira separada ao *CAS*. Com isso, outros sistemas poderão utilizar o mecanismo de representação de expressões.

A ferramenta criada, denominada *JEzMath*, foi registrada sob a licença *LGPL* [Fre07b], que é totalmente compatível com a *GPL* utilizada no *Zeno*. Sendo assim, é possível modificar a ferramenta, desde que as restrições da licença sejam respeitadas. Totalmente implementada em Java, a ferramenta pode ser utilizada por qualquer aplicação escrita na linguagem. A sintaxe utilizada para criação de expressões é um subconjunto de *LaTeX*, linguagem bastante conhecida no meio acadêmico. As propriedades de exibição podem ser alteradas e pode-se selecionar qualquer parte de uma expressão. Todos os requisitos exigidos são suportados pela ferramenta desenvolvida. A Figura 5.6 mostra algumas expressões exibidas pelo mecanismo de representação.

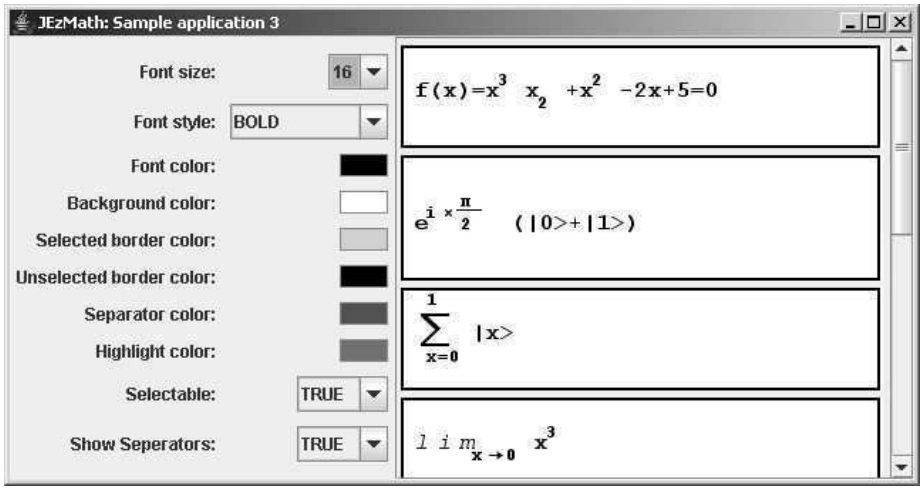


Figura 5.6: Exemplo de representação de expressões no mecanismo de representação adotado no *CAS*

O mecanismo de representação de expressões emprega outros componentes em sua implementação, são eles:

- *JLex* - é um gerador de analisador léxico desenvolvido em Java. Utilizado para criar o *scanner* que traduz cada *token* de entrada (*LaTeX*) para o *token* de saída (*Unicode*) correspondente.

- *Java Cup* - é um gerador de *parsers* desenvolvido em Java. Utilizado criar o *parser* responsável por checar e traduzir a palavra de entrada (*LaTeX*) para um conjunto de *tokens* (*Unicode*) com suas respectivas propriedades.

O funcionamento do *JEzMath* consiste em traduzir a palavra de entrada, em sintaxe *LaTeX*, para uma palavra composta de caracteres *unicode*. Cada *token* de entrada irá possuir um conjunto de propriedades, as quais serão utilizadas para determinar as configurações de exibição do mesmo. As propriedades podem determinar, por exemplo, se um *token* é o numerador ou denominador de uma fração, ou se este *token* é um expoente. A gramática apresentada a seguir, ilustrada através de uma simplificação da sintaxe do *Java Cup*, descreve o subconjunto da linguagem *LaTeX* suportado pelo mecanismo de representação.

Código Fonte- Gramática subconjunto da linguagem LaTeX

```
1  expression     ::= terms expression
2                   | terms ;
3  terms          ::= symbol
4                   | bigsymbol
5                   | italicsymbol
6                   | alpha
7                   | number
8                   | subscriptterm
9                   | supscriptterm
10                  | fracterm
11                  | rootterm
12                  | overterm
13                  | underterm ;
14 subscriptterm ::= subscript { expression } ;
15 supscriptterm ::= supscript { expression } ;
16 fracterm       ::= frac { expression } { expression } ;
17 rootterm       ::= sqrt { expression }
18                  | sqrt [ index ] { expression } ;
19 index          ::= alpha
20                  | number ;
21 overterm       ::= overline { expression }
22                  | widehat { expression }
23                  | widetilde { expression }
24                  | overrightarrow { expression }
25                  | overleftarrow { expression } ;
26 underterm      ::= underline { expression } ;
27 digit          ::= [0-9]
28 number         ::= {digit}+
```

```
29 char          ::= [a-z A-Z]
30 alpha         ::= {char}+
```

A Figura 5.7 exibe o modelo conceitual do mecanismo de representação.

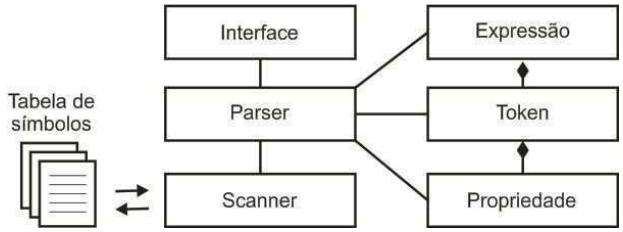


Figura 5.7: Modelo conceitual do mecanismo de representação do *CAS*

A Figura 5.8 ilustra o funcionamento interno do *JEzMath*. A *string LaTeX* fornecida como entrada corresponde a expressão $\frac{1}{2}(|0\rangle \otimes |0\rangle)$.

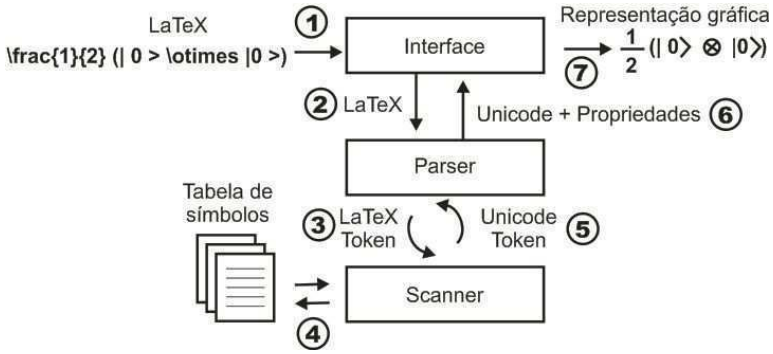


Figura 5.8: Funcionamento interno do mecanismo de representação

1. A *string LaTeX* é fornecida para interface;
2. A interface entrega a *string* para o *parser*;
3. O *parser* verifica se a sintaxe é obedecida e então solicita ao *scanner* a tradução de cada *token* de *LaTeX* para sua representação correspondente em *Unicode*;
4. O *scanner* consulta a tabela de símbolos para identificar qual *token Unicode* corresponde ao *token LaTeX* fornecido (e.g. \otimes → \u2297);
5. O *scanner* devolve o *token Unicode* correspondente ao *token LaTeX* solicitado;

6. O *parser* adiciona as propriedades pertencentes ao *token Unicode* e devolve uma expressão, a qual consiste em um conjunto de *tokens* e suas propriedades (e.g $1 \rightarrow numerador, 2 \rightarrow denominador$);

7. A interface retorna a expressão em sua forma de representação gráfica.

5.2 O mecanismo de manipulação de expressões

Para realizar as manipulações simbólicas foi necessário criar uma forma de representação interna que suportasse a execução de cálculos algébricos. Os resultados numéricos obtidos no simulador não possuem nenhum tipo de ligação com os resultados simbólicos. A representação interna dos "objetos" matemáticos suportados no *CAS* faz uso da "estrutura" representada no pacote *zeno.cas.symbolic.math*. Um diagrama de classes simplificado deste pacote é apresentado na Figura 5.9.

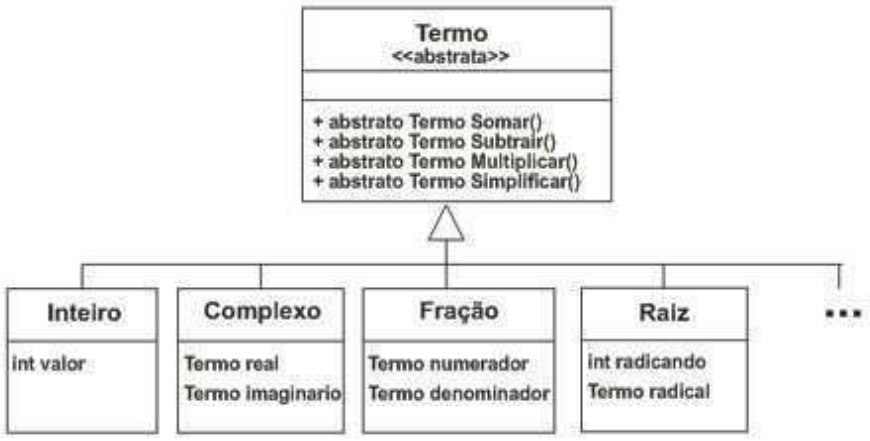


Figura 5.9: Diagrama de classes simplificado do pacote *zeno.cas.symbolic.math*

As classes existentes no pacote citado permitem a realização dos cálculos simbólicos relacionados ao domínio de circuitos quânticos. A definição de cada elemento existente na estrutura possibilita um encadeamento de estruturas. Para ilustrar a afirmação, a Figura 5.10 mostra um encadeamento para representação de $\frac{1}{\sqrt{2^2}}$.

Cada uma das classes pertencentes ao pacote implementa as operações de soma, subtração e multiplicação, além da ação de simplificar uma representação (e.g. $\frac{2}{4} \rightarrow \frac{1}{2}$). A

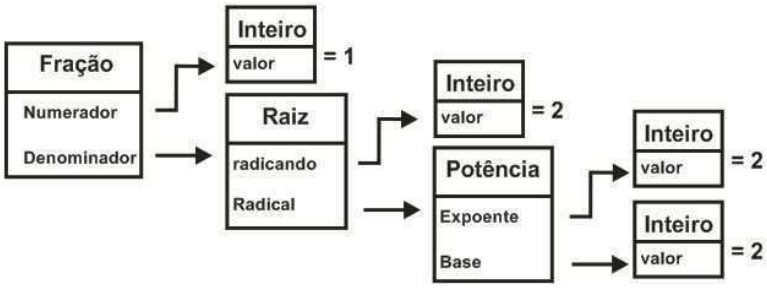


Figura 5.10: Encadeamento para representação de $\frac{1}{\sqrt{2^2}}$

operação de divisão é interpretada como uma fração, por isso, não é implementada por nenhuma das classes.

Além das estruturas matemáticas citadas, também foram criadas representações internas de estados e portas quânticas. Estas permitem que as diversas manipulações e transformações sejam realizadas. É possível, por exemplo, exibir estados na base decimal ou usar a notação de Dirac para exibir uma porta quântica.

Uma expressão simbólica no *CAS* corresponde a um conjunto de "objetos" matemáticos, um conjunto de portas e um conjunto de estados quânticos. Durante qualquer simulação, uma forma de representação padrão é utilizada na expressão. As diversas manipulações podem alterar internamente um estado quântico ou modificar a maneira como uma expressão será exibida. Por exemplo, a aplicação de uma porta irá alterar o(s) estado(s) onde ela é aplicada, por sua vez, a mudança na base de representação altera apenas a forma como cada estado pertencente à expressão será exibido, mas não o modifica internamente. A Figura 5.11 mostra o modelo conceitual do mecanismo de manipulação.

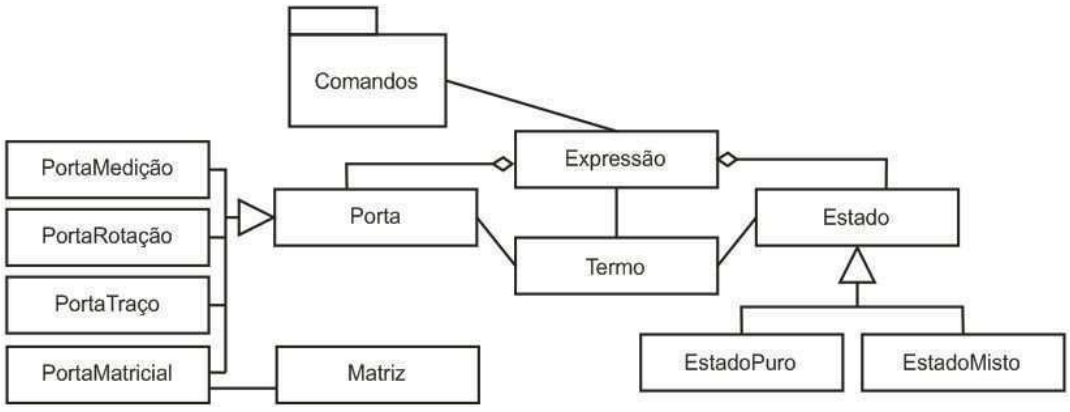


Figura 5.11: Modelo conceitual do mecanismo de manipulação.

As manipulações suportadas no sistema são encontradas no pacote *zeno.cas.commands*. Existem manipulações que realizam a aplicação de portas aos seus respectivos qubits, outras que contraem a representação de um registrador exibindo o mesmo através de somatórios.

Uma das principais manipulações fornecidas por um *CAS* é a execução de simplificações. Como não existe uma definição formal do que é mais simples [BL06] [Car04] [Mos71], os diversos sistemas existentes adotam diferentes definições.

A subjetividade existente no que vem a ser uma simplificação torna difícil a criação de uma definição amplamente aceita, uma representação é mais simples do que outra dependendo do contexto em que será utilizada.

Para o *Zeno* uma simplificação é definida como uma aplicação conjunta de comandos. Uma simplificação cria uma árvore de expressões onde todas as possíveis representações estão contidas. O usuário pode então definir qual expressão deverá ser utilizada dentro do contexto trabalhado.

Para realizar uma simplificação, os comandos disponíveis no *CAS* são classificados em quatro grupos. O primeiro identifica os comandos que não serão utilizados. Os grupos restantes, descritos a seguir, são usados de três diferentes maneiras:

- Grupo inicial - são aplicados sobre a expressão original. Apenas uma expressão é obtida após a execução de todos os comandos deste grupo;
- Grupo ramificável - são aplicados sobre o conjunto de expressões obtido na execução do grupo de comandos anterior (inicial ou ramificável). Geram várias diferentes expressões sobre as quais é possível aplicar novos comandos deste grupo;
- Grupo final - são aplicados sobre as expressões resultantes após a execução do grupo ramificável.

A Figura 5.12 exibe a árvore de simplificação para a expressão $X|0\rangle I|0\rangle$. Apenas as expressões finais identificadas pelas letras A, B, C, D e E são exibidas para o usuário, que pode então selecionar qual a representação desejada.

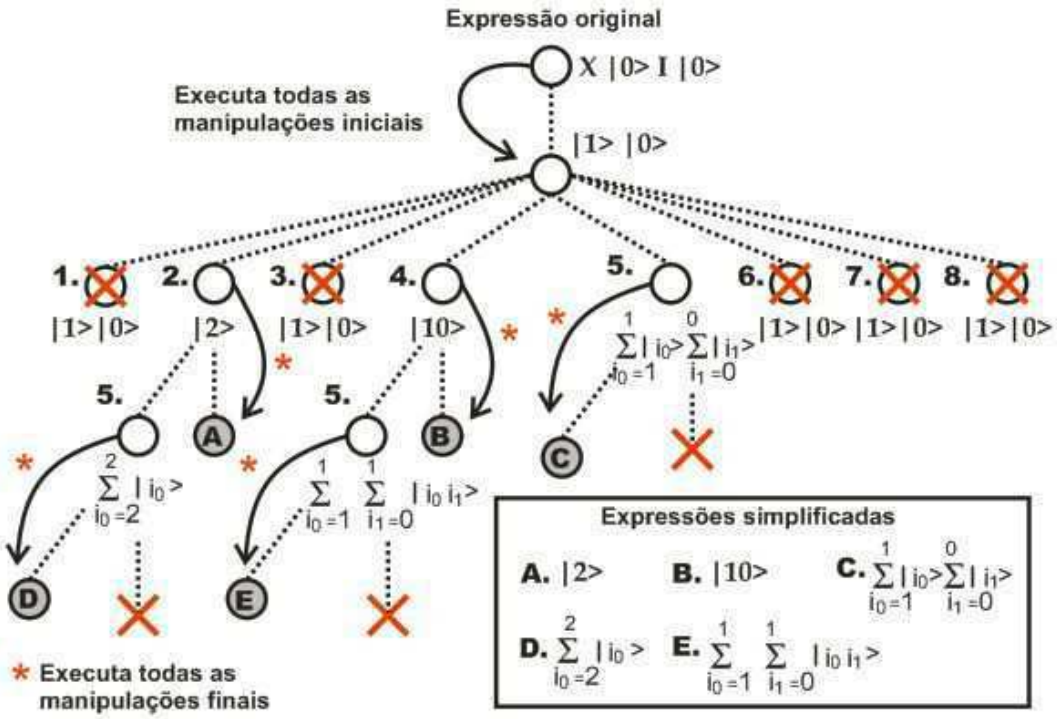


Figura 5.12: Exemplo de árvore de simplificação.

5.3 Uma visão geral do novo *Zeno*

As seções anteriores descreveram os mecanismos de representação e manipulação de expressões, os quais compõem o *CAS* do simulador *Zeno*. Esta seção fornece uma visão geral de como estes mecanismos trabalham juntos, ou seja, descreve-se aqui o funcionamento interno do *CAS*. A arquitetura do simulador como um todo também será apresentada.

A Figura 5.13 exibe a arquitetura do *CAS*. Pode-se observar que o mecanismo de manipulação está contido na aplicação, já o mecanismo de representação é um componente separado.

Figura 5.13: Arquitetura do *CAS*.

A Figura 5.14 ilustra o funcionamento interno do *CAS*. A descrição feita a seguir refere-se à execução de dois comandos. O primeiro, rotulado como “A”, é a solicitação de uma

modificação na base de representação de binário para decimal. O segundo, rotulado como "B", corresponde à execução de um passo de simulação, no qual uma porta Hadamard é inserida.

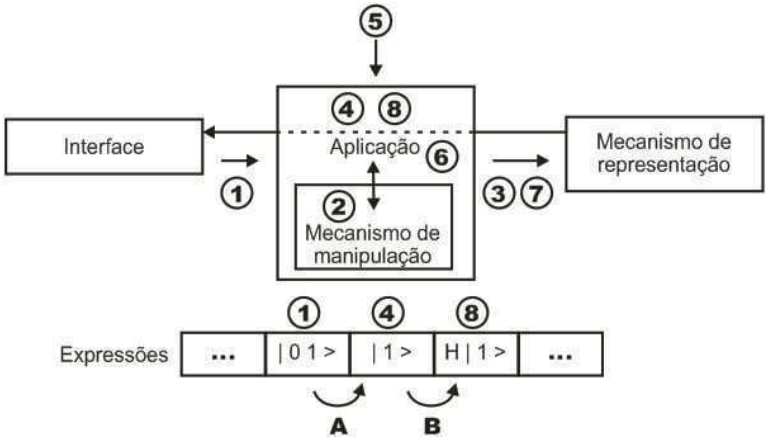


Figura 5.14: Funcionamento interno do mecanismo de manipulação.

1. A - a última expressão existente no *CAS* corresponde a representação do estado $|01\rangle$. O usuário então solicita, a partir da interface do *CAS*, que a base decimal seja utilizada;

2. A - a aplicação recebe a solicitação e utilizando o mecanismo de manipulação realiza a alteração na base, criando assim uma nova expressão;

3. A - a aplicação solicita ao mecanismo de representação a forma de exibição da nova expressão;

4. A - o mecanismo entrega a representação gráfica da expressão para a aplicação, a qual, por sua vez, realiza a inserção da expressão na interface;

5. B - o usuário executa um passo de simulação, o qual corresponde a inserção da porta Hadamard. O editor de circuitos envia para o *CAS* a informação necessário para realizar a ação;

6. B - a aplicação gera uma nova expressão que contém a porta Hadamard;

7. B - a aplicação solicita ao mecanismo de representação a forma de exibição da nova expressão;

8. B - o mecanismo então entrega a representação gráfica da expressão para a aplicação que realiza a inserção desta na interface.

Descrito o funcionamento interno do *CAS*, pode-se então exibir a nova arquitetura do simulador *Zeno*. A Figura 5.15 mostra esta arquitetura.

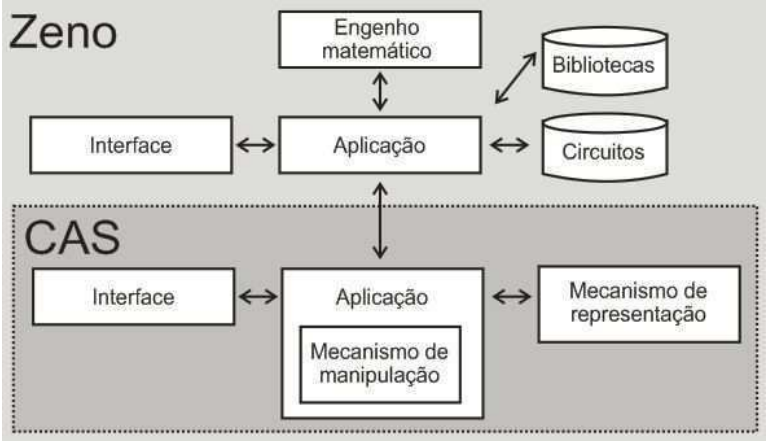


Figura 5.15: Arquitetura do *Zeno*.

Nenhum aspecto da arquitetura do simulador original foi modificado. Apenas funcionalidades de comunicação, do editor de circuitos com o *CAS*, foram adicionadas. Esta comunicação é realizada de uma camada de aplicação para outra camada de aplicação.

O usuário pode escolher qualquer interface para realizar determinadas ações, tais como, executar os diversos tipos de simulação, abrir e salvar circuitos, entre outras. Ações pertencentes somente ao editor devem ser executadas em sua interface, por exemplo, inserir portas, editar o estado inicial. Do mesmo modo, ações relacionadas somente ao *CAS* devem ser executadas a partir de sua interface, tais como, solicitar execução de manipulação, alterar configurações de exibição.

Capítulo 6

Análise de resultados

Neste capítulo, serão apresentados exemplos que ilustram as funcionalidades simbólicas desenvolvidas, exibindo assim os resultados obtidos.

Para demonstrar que não houve perda de desempenho, os testes apresentados em [Cab04] foram novamente realizados. Eles foram implementados e executados no simulador *Zeno* em sua versão original e na nova versão. Os testes realizados na nova versão do *Zeno* foram primeiramente executados sem o apoio do *CAS* e, em um segundo momento, foram realizados após a inicialização deste.

Dois computadores foram utilizados na realização dos testes, estando a configuração destas máquinas descrita na tabela seguinte.

Tabela 6.1: Configuração das máquinas utilizadas na realização dos testes.

	PC 1	PC 2
Processador	AMD Sempron 2800+ 1,6 Ghz	AMD Duron 800 MHz
Memória RAM	512 MB	128 MB

A estratégia usada na realização dos testes foi a seguinte:

1. Projetar os circuitos nos simuladores *Zeno* original e novo *Zeno*;
2. Executar os circuitos nos simuladores *Zeno* original e novo *Zeno*;
3. Comparar tempo de execução e os resultados obtidos;
4. Iniciar o *CAS* e executar os circuitos no novo *Zeno*;

5. Comparar o tempo de execução obtido antes e depois de iniciar o *CAS*.

6.1 Somador de dois bits

Trata-se de um circuito que realiza a soma de dois números de dois bits, resultando em um número de três bits. A Figura 6.1 exibe o circuito implementado no novo *Zeno*. Observa-se que o circuito possui um registrador de sete qubits e que são empregadas nove portas controladas na execução da soma.

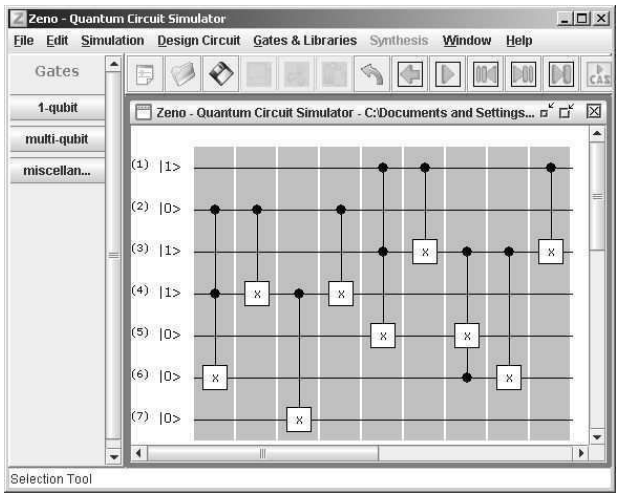


Figura 6.1: Circuito somador implementado no simulador *Zeno*

A Tabela 6.2 exibe os tempos de execução obtidos em cada um dos simuladores. Os valores apresentados consistem na média de tempo obtido para execução de cada uma das possíveis entradas.

Tabela 6.2: Tempo de execução do circuito somador de dois bits.

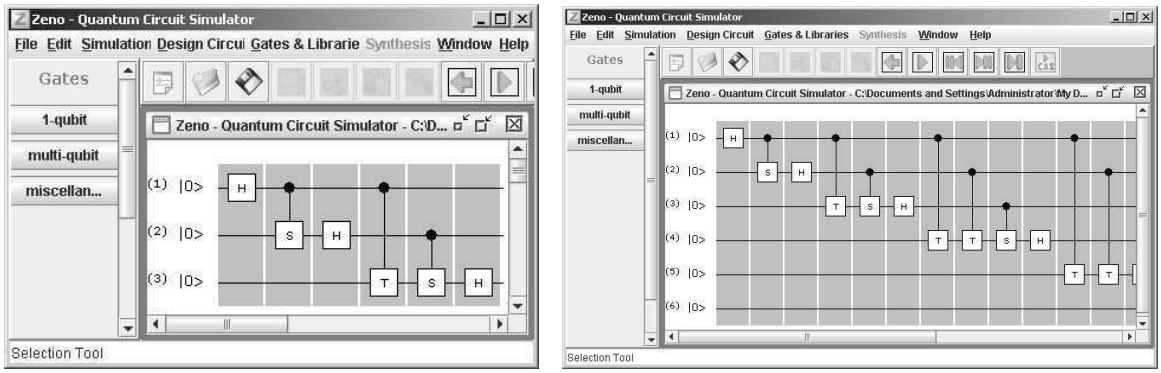
Simulador	tempo (puro) PC 1	tempo (misto) PC 1	tempo (puro) PC 2	tempo (misto) PC 2
Zeno original	63ms	4s 600ms	660ms	22s 690ms
novo *Zeno*	47ms	3s 500ms	660ms	15s 320ms
novo *Zeno* + *CAS*	47ms	3s 500ms	660ms	15s 320ms

Os resultados obtidos mostram que o tempo de execução nos simuladores é compatível. Ambos simularam o circuito em milisegundos para estados puros, e segundos, para estados

mistos. A nova versão do *Zeno* apresentou um desempenho ligeiramente melhor que a versão original. Acredita-se que esta melhoria tenha ocorrido devido à atualização do engenho matemático *J.A.D.E*, que foi substituído por uma versão mais recente denominada *JScience*.

6.2 Transformada de Fourier quântica

O algoritmo foi testado em duas implementações: a primeira utilizando três qubits e seis portas, e a segunda empregando seis qubits e vinte e uma portas. As Figuras 6.2(a) e 6.2(b) mostram circuitos criados no simulador *Zeno* implementando a *QFT* para três e seis qubits respectivamente.



(a) Circuito *QFT* para três qubits implementado no simulador *Zeno*.

(b) Circuito *QFT* para seis qubits implementado no simulador *Zeno*.

Figura 6.2: Circuitos implementando a *QFT*.

A Tabela 6.3 exibe os tempos de execução obtidos em cada um dos simuladores no circuito *QFT* para três qubits. Os resultados obtidos mostram que o tempo de execução nos simuladores é compatível, pois todos simularam o circuito de maneira "imediata".

A Tabela 6.4 exibe os tempos de execução obtidos em cada um dos simuladores no circuito *QFT* para seis qubits.

Os resultados obtidos mostram que o tempo de execução nos simuladores é compatível, uma vez que todos simularam o circuito em uma ordem de tempo similar. Mais uma vez, a nova versão do *Zeno* apresentou um desempenho ligeiramente melhor que a versão original.

Tabela 6.3: Tempo de execução do circuito *QFT* para três qubits.

Simulador	tempo (puro) PC 1	tempo (misto) PC 1	tempo (puro) PC 2	tempo (misto) PC 2
Zeno original	< 1ms	< 1ms	< 1ms	< 1ms
novo *Zeno*	< 1ms	< 1ms	< 1ms	< 1ms
novo *Zeno* + *CAS*	< 1ms	< 1ms	< 1ms	< 1ms

Tabela 6.4: Tempo de execução do circuito *QFT* para seis qubits.

Simulador	tempo (puro) PC 1	tempo (misto) PC 1	tempo (puro) PC 2	tempo (misto) PC 2
Zeno original	93ms	906ms	330ms	3s 900ms
novo *Zeno*	78ms	660ms	330ms	2s 690ms
novo *Zeno* + *CAS*	78ms	660ms	330ms	2s 690ms

6.3 Circuito para calcular o autovalor

O circuito para calcular o autovalor de um operador quântico faz uso de quarenta e cinco portas. A Figura 6.3 exibe a implementação deste circuito no simulador *Zeno*.

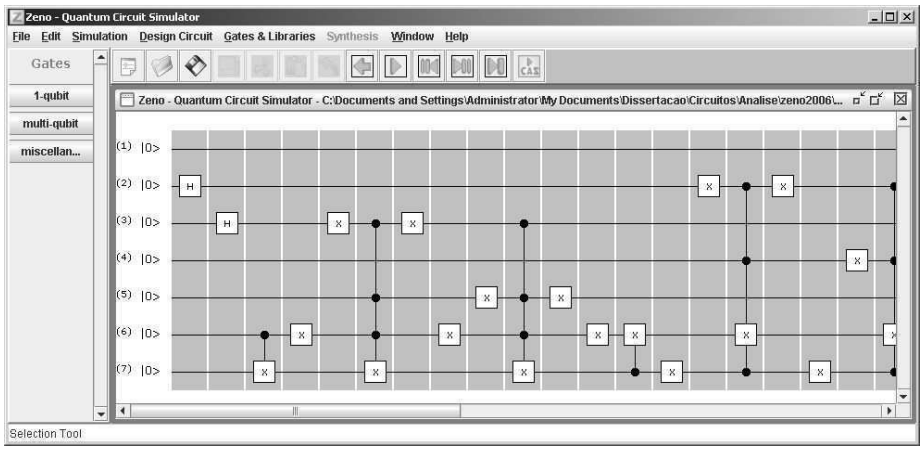


Figura 6.3: Circuito para cálculo de autovalor implementado no simulador *Zeno*

A Tabela 6.5 exibe os tempos de execução obtidos em cada um dos simuladores no circuito para o cálculo de autovalores.

Na simulação deste circuito, mais uma vez, o desempenho da nova versão foi ligeiramente melhor do que na versão original do simulador.

Tabela 6.5: Tempo de execução do circuito para calcular o autovalor de um operador quântico.

Simulador	tempo (puro) PC 1	tempo (misto) PC 1	tempo (puro) PC 2	tempo (misto) PC 2
Zeno original	17s 250ms	41s 750ms	1m 3s 500ms	2m 50s 390ms
novo *Zeno*	13s 600ms	30s 200ms	56s 500ms	2m 3s 200ms
novo *Zeno* + *CAS*	13s 600ms	30s 200ms	56s 500ms	2m 3s 200ms

6.4 Algoritmo de Deutsch

Para ilustrar o uso das manipulações suportadas pelo simulador, serão exibidos alguns exemplos, executados sobre as descrições de estado obtidas em passos intermediários do algoritmo de Deutsch. Esta ilustração apresenta também o uso conjunto da representação gráfica do circuito e da descrição simbólica do estado do sistema.

O algoritmo de Deutsch determina em uma única execução se uma função $f : \{0, 1\} \rightarrow \{1, 0\}$ é balanceada, $f(0) \neq f(1)$, ou constante, $f(0) = f(1)$. O último passo do algoritmo consiste em realizar uma medição sobre o primeiro qubit, verificando se a função testada é balanceada ($\pm |1\rangle$) ou constante ($\pm |0\rangle$).

A Figura 6.4 exibe um circuito implementando o algoritmo de Deutsch para a função soma módulo dois.

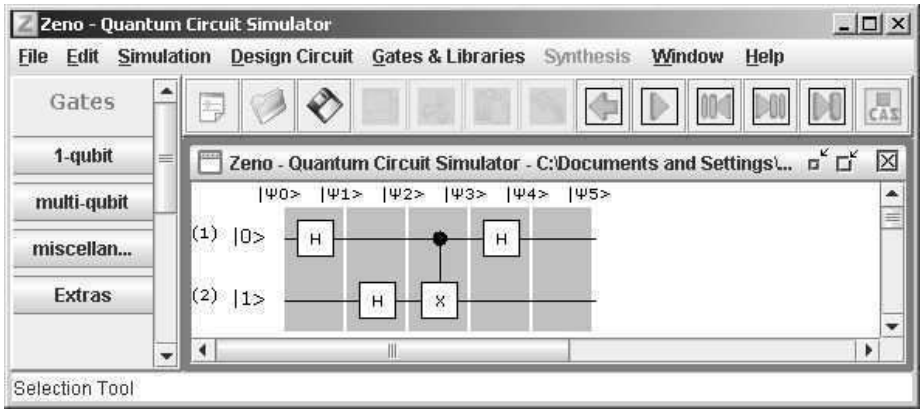
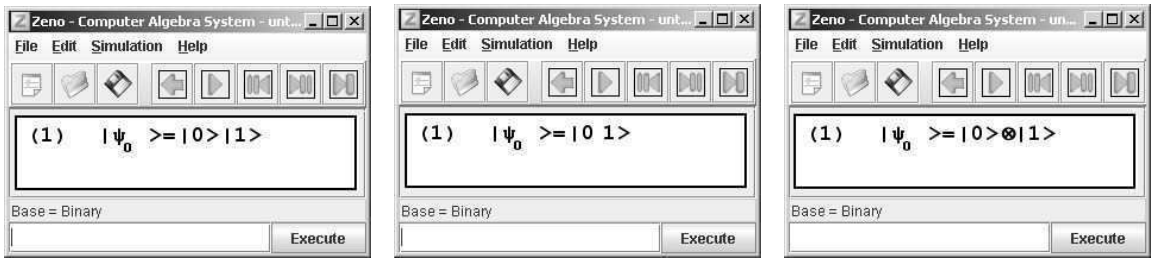


Figura 6.4: Circuito implementando o algoritmo de Deutsch.

O estado inicial $|\psi_0\rangle$ pode ser representado através de qualquer uma das expressões exibidas na Figura 6.5. O usuário pode determinar a forma de representação alterando a confi-

guração do sistema.



(a) Estado $|\psi_0\rangle$ exibido com uma configuração alternativa.

(b) Estado $|\psi_0\rangle$ exibido com a configuração padrão.

(c) Estado $|\psi_0\rangle$ exibido com uma configuração alternativa e o símbolo tensorial.

Figura 6.5: Alternativas de representações do estado $|\psi_0\rangle$ no *Zeno*.

Continuando a execução do algoritmo através de passos de simulação, o estado $|\psi_2\rangle$ pode ser obtido. Este estado corresponde à aplicação dos operadores Hadamard sobre os qubits de entrada, ele pode ser representado através das expressões exibidas na Figura 6.6.

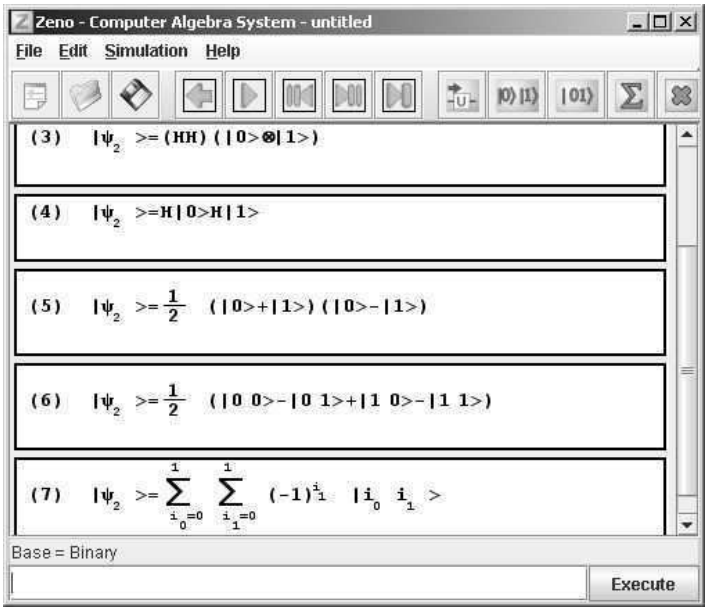


Figura 6.6: Representações do estado $|\psi_2\rangle$.

Combinando passos de simulação e manipulações é possível obter diversas descrições para os estados intermediários. O estado $|\psi_3\rangle$ pode ser descrito por qualquer uma das expressões exibidas na Figura 6.7.

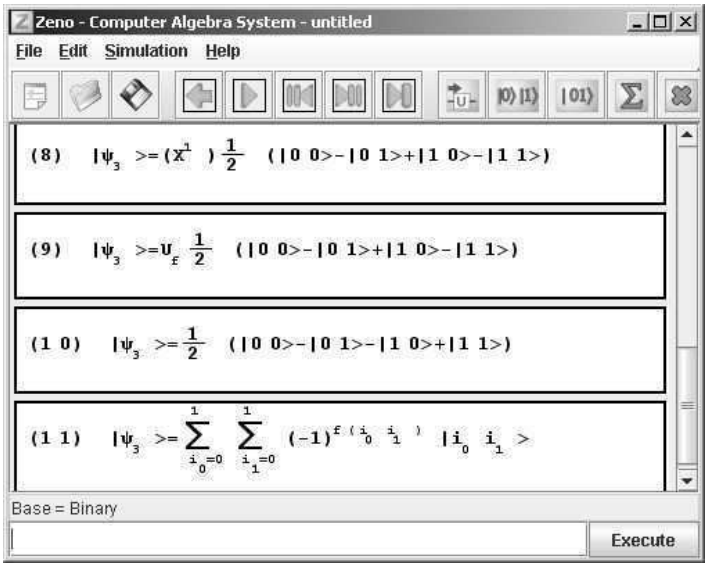


Figura 6.7: Representações do estado $|\psi_3\rangle$.

Finalmente, no estado $|\psi_4\rangle$ é aplicado novamente o operador Hadamard sobre o primeiro qubit, levando a superposição a um estado da base canônica. O estado $|\psi_4\rangle$ pode ser descrito por qualquer uma das expressões exibidas na Figura 6.8.

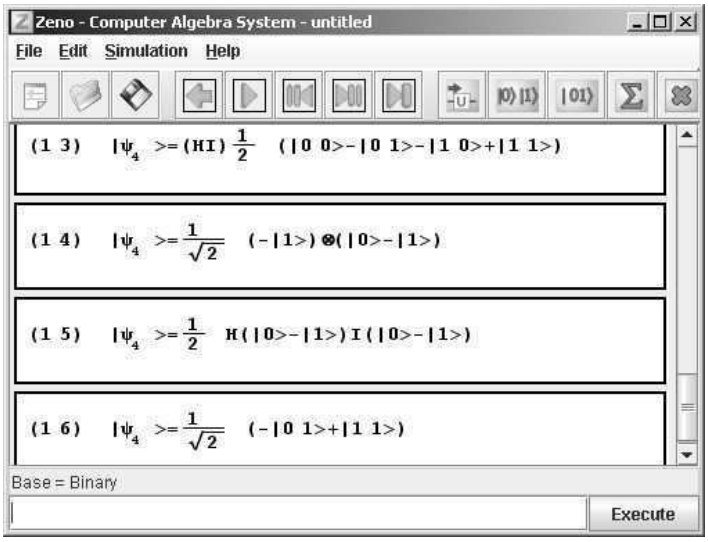


Figura 6.8: Representações do estado $|\psi_4\rangle$.

Pode-se concluir que a função computada no exemplo é balanceada, pois, observa-se que o primeiro qubit corresponde ao estado $|1\rangle$.

6.5 Análise de desempenho (*Benchmark*)

Com o intuito de identificar os limites em relação à quantidade de qubits, empregados em uma simulação, foi feita uma análise de desempenho sobre a nova versão do *Zeno*. Por se tratar de uma aplicação Java, o desempenho do simulador é bastante relacionado às configurações da máquina virtual (*JVM*) utilizada. O *PC1*, descrito anteriormente, foi utilizado para realização dos testes.

Por padrão são alocados 64MB para execução do simulador. Os testes foram primeiramente executados neste perfil. Em um segundo momento, a quantidade de memória padrão adotada foi alterada para o limite de 512MB, sendo os testes novamente executados.

Com isso, obteve-se que, com a configuração padrão, circuitos com no máximo 19 qubits, sem portas controladas, e 10 qubits, com portas controladas, são executados em um tempo inferior a 5 minutos. Alterando-se o limite para 512MB, circuitos com no máximo 21 qubits, sem portas controladas, e 11 qubits, com portas controladas, são executados em um tempo inferior a 5 minutos. Uma tabela detalhada descrevendo os circuitos e resultados obtidos é apresentada no Apêndice C.

6.6 Análise dos resultados

Após a execução dos testes descritos, pode-se concluir que a descrição simbólica fornecida pelo *CAS* é correta, uma vez que corresponde à descrição numérica original. Pode-se confirmar também que o desempenho do simulador não é afetado com a execução do *CAS*, visto que os tempos de simulação dos circuitos não foram modificados após a inicialização do sistema.

Os exemplos ilustrados mostram que a descrição fornecida pelo *Zeno* é fiel àquelas encontradas na literatura. Pode-se concluir que, com a execução de passos de simulação e de manipulações, é possível obter a descrição mais adequada de um estado para o contexto desejado. Com isso, o estudo e a pesquisa de quaisquer algoritmos podem ser facilitados.

Capítulo 7

Conclusões e trabalhos futuros

Neste capítulo, são apresentadas as considerações finais relacionadas ao trabalho, bem como, são descritos alguns trabalhos futuros que poderão ser desenvolvidos para contribuir com a evolução do simulador.

7.1 Conclusões

Este trabalho descreveu o desenvolvimento de um *CAS* específico para o contexto de circuitos quânticos. O *CAS* foi criado como uma extensão do simulador *Zeno*, fazendo com que este forneça a representação gráfica do circuito aliada à descrição simbólica do estado do sistema.

Com a conclusão deste trabalho, tem-se que o *Zeno* é hoje o único simulador de circuitos quânticos capaz de fornecer, em uma única ferramenta, a descrição de um algoritmo de maneira fiel àquelas encontradas na literatura.

Para realizar a implementação da extensão, fez-se necessário efetuar uma série de atualizações e modificações sobre o simulador original. Nenhuma destas mudanças afetou o projeto original do *Zeno* e em alguns casos, a execução de funcionalidades foi alterada. O conjunto de testes originais continha 168 casos de teste, hoje existem mais de 650 casos de testes implementados.

Os principais componentes desenvolvidos para o *CAS* foram o mecanismo de representação de expressões, denominado *JEzMath*, e o mecanismo de manipulação de expressões. O *JEzMath* foi criado como um componente separado do *CAS* e pode ser usado por outras aplicações. O mecanismo de manipulações é parte integrante do simulador, sua utilização

está bastante ligada ao *Zeno*, dessa forma, ele não pode ser utilizado por outras aplicações.

O simulador facilita a investigação do algoritmo simulado, de modo que o usuário pode trabalhar de forma muito mais rápida e eficiente utilizando o simulador do que realizando manipulações matemáticas com papel e caneta.

Espera-se que o simulador *Zeno* se torne uma ferramenta ainda mais valiosa nos contextos de pesquisa e ensino. Para pesquisa, a ferramenta deverá facilitar o desenvolvimento de algoritmos, simplificando a investigação do "objeto" estudado. No contexto de ensino, tanto professores quanto alunos deverão se beneficiar com a extensão, haja vista que, os professores terão mais uma ferramenta didática que os ajudará a ilustrar os conceitos trabalhados em sala, enquanto, os alunos poderão utilizar o simulador para auxiliar sua busca pela solução de exercícios ou na confirmação dos resultados obtidos.

O simulador se tornou também um instrumento de divulgação, em nível nacional e internacional, do grupo de pesquisas em computação e informação quântica da Universidade Federal de Campina Grande, visto que foram publicados artigos científicos em eventos nacionais e internacionais ligados à computação quântica.

7.2 Trabalhos futuros

O simulador *Zeno* é hoje um dos simuladores de circuitos quânticos que merecem destaque, pois possui um rico conjunto de funcionalidades e é o único simulador a fornecer uma descrição completa da linguagem de circuitos. A qualidade do simulador é um motivo de incentivo para desenvolver trabalhos que venham a enriquecer a ferramenta.

A única funcionalidade planejada para o *CAS* que ainda não foi implementada, consiste na possibilidade de aplicar comandos sobre determinadas partes de uma expressão. Tal característica será desenvolvida em breve, permitindo que uma variedade ainda maior de manipulações seja realizada.

Os cálculos numéricos realizados em uma simulação consistem basicamente em realizar operações sobre matrizes e vetores. Atualmente, a representação de tais entidades e a execução das operações é realizada através do uso do componente *JScience*. Neste, a representação de matrizes e vetores é realizada em sua forma densa. Recursos (tempo e memória) poderiam ser melhor utilizados se apenas os elementos não nulos fossem armazenados. Visto isso, é

interessante substituir ou atualizar o componente para que sejam utilizadas matrizes esparsas na execução das operações internas do simulador. Serão avaliadas as alternativas atuais e, caso nenhuma seja adequada, um novo componente será implementado.

Algumas das funcionalidades planejadas para a versão original do simulador, tais como a simulação de erros ou a simplificação de circuitos, não foram desenvolvidas até o momento. Pode-se então implementar as funcionalidades, além de realizar a execução de tarefas de perfilamento, com o intuito de descobrir o(s) gargalo(s) que compromete(m) o desempenho do simulador na execução do circuito de cálculo de autovalores.

O código do simulador pode ser refatorado para que as melhorias criadas para a versão 6 da *J2SE* sejam exploradas. Esta atividade é importante para permitir que as futuras modificações ou a implementação de novas funcionalidades sejam realizadas facilmente.

Para facilitar o uso do *Zeno* é importante criar um manual de consulta onde os usuários possam esclarecer quaisquer dúvidas que ocorram durante a utilização do simulador. Este manual deverá ser disponibilizado em um relatório técnico e como forma de ajuda dentro do próprio sistema.

O conjunto de manipulações e comandos fornecidos atualmente pode ser ampliado e, com isso, pode-se enriquecer as possibilidades de investigação dos algoritmos simulados no *Zeno*.

Bibliografia

[Apa06] Apache Software Foundation. The Apache ANT Project. `http://ant.apache.org/`, 2006. Acessado em março de 2007.

[BL06] Alexandre Andrade Barbosa & Bernardo Lula. Simplificação automática de expressões matemáticas. Relatório Técnico, Universidade Federal de Campina Grande - UFCG, 2006.

[BLL06] Alexandre Andrade Barbosa, Bernardo Lula, & Aércio Ferreira Lima. Zeno Documentação: Análise técnica sobre o simulador. Relatório Técnico, Universidade Federal de Campina Grande - UFCG, 2006.

[Boh02] Mark Bohr. Intel's 90 nm Technology: Moore's Law and More. `ftp://download.intel.com/technology/silicon/Bohr_IDF_0902.pdf`, 2002. Acessado em março de 2007.

[Boh05] Mark Bohr. 65 nm Technology for High Performance and Low Power. `ftp://download.intel.com/technology/silicon/IDF_Aug_05_65nm_logic.pdf`, 2005. Acessado em março de 2007.

[Bos07] Bert Bos. W3C Math Home. `http://www.w3.org/Math/`, 2007. Acessado em março de 2007.

[Bro24] Louis Broglie. *Recherches sur la théorie des quanta (tr. Pesquisas sobre a teoria quântica)*. Tese de doutorado, Faculdade de Ciências Universidade de Paris, 1924.

[Bur05] Oliver Burn. Checkstyle 4.1. `http://checkstyle.sourceforge.net/`, 2005. Acessado em março de 2007.

[BV93] E. Bernstein & Umesh Vazirani. Quantum Complexity Theory. In *Symposium on Theory of Computation, ACM, New York*, pages 11–20, 1993.

[Cab04] Gustavo Elálio Cabral. Uma ferramenta para simulação de circuitos quânticos. Dissertação de mestrado, Universidade Federal de Campina Grande (UFCG), 2004.

[Car04] Jacques Carette. Understanding Expression Simplification. In *ISSAC 2004*, pages 72–79, 2004.

[CGK98] I. L. Chuang, N. A. Gershenfeld, & M. Kubinec. Experimental Implementation of Fast Quantum Searching. *Phisical Review letters*, 80(15):3408–3411, 1998.

[CL03] Gustavo Elálio Cabral & Bernardo Lula. O estado da arte em ferramentas para síntese e simulação de circuitos quânticos. Relatório Técnico, Universidade Federal de Campina Grande - UFCG, 2003.

[CLL05] Gustavo Eulálio Cabral, Bernardo Lula, & Aércio Ferreira Lima. ZENO: a new graphical tool for design and simulation of quantum circuits. *Defense and Security Symposium 2005 - Quantum Information and Computation III*, 5815:127–137, 2005.

[D-W07] D-Wave. World's First Commercial Quantum Computer Demonstrated. `http://www.dwavesys.com/index.php?mact=News,cntnt01,detail,0&cntnt01articleid=4&cntnt01origid=15&cntnt01returnid=21`, 2007. Acessado em março de 2007.

[Dau06a] Jean-Marie Dautelle. jade. `https://jade.dev.java.net/`, 2006. Acessado em março de 2007.

[Dau06b] Jean-Marie Dautelle. Javolution. `http://javolution.org/`, 2006. Acessado em março de 2007.

[Dau06c] Jean-Marie Dautelle. JScience. `http://jscience.org/`, 2006. Acessado em março de 2007.

[Dav05] J.H. Davenport. Computer Algebra - Past, Present and Future. `http://citeseer.ist.psu.edu/96199.html`, 2005. Acessado em março de 2007.

[Deu85] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society, London, Series A*, 400:97–117, 1985.

[Deu89] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society, London, Series A*, 425:73–90, 1989.

[Dol06] Mark Doliner. Cobertura. `http://cobertura.sourceforge.net/`, 2006. Acessado em março de 2007.

[Fat04] Richard J. Fateman. How Can We Speak Math? `dlp.cs.berkeley.edu/~fateman/papers/speakmath.pdf`, 2004. Acessado em março de 2007.

[Fey82] Richard Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[Fra05] Michael P. Frank. Approaching the Physical Limits of Computing. In *ISMVL*, pages 168–185, 2005.

[Fre07a] Free Software Foundation. GNU General Public License. `http://www.gnu.org/copyleft/gpl.html`, 2007. Acessado em março de 2007.

[Fre07b] Free Software Foundation. GNU Lesser General Public License. `http://www.gnu.org/licenses/lgpl.html`, 2007. Acessado em março de 2007.

[Gay05] Todd Gayley. A MathLink Tutorial. `http://library.wolfram.com/infocenter/TechNotes/174/`, 2005. Acessado em março de 2007.

[GM06] David Gilbert & Thomas Morgner. JFreeChart. `http://www.jfree.org/jfreechart/index.php`, 2006. Acessado em março de 2007.

[GQK04] Erich Gamma, Jochen Quante, & Wolfram Kaiser. JHotDraw as Open-Source Project. `http://www.jhotdraw.org/`, 2004. Acessado em março de 2007.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC 1996*, pages 212–219, 1996.

[IQU07] IQUANTA. Zeno - Quantum Circuit Simulator. `http://dsc.ufcg.edu.br/~iquanta/zeno/`, 2007. Acessado em março de 2007.

[Jip05] Peter Jipsen. ASCIIMathML. `http://www1.chapman.edu/~jipsen/mathml/asciimath.html`, 2005. Acessado em março de 2007.

[JM98] J. A. Jones & M. Mosca. Implementation of a Quantum Algorithm to Solve Deutsch's Problem on a Nuclear Magnetic Resonance Quantum Computer. *J. Chem. Phys.*, 109:1648–1653, 1998.

[JMH98] J. A. Jones, M. Mosca, & R. H. Hansen. Implementation of a Quantum Search Algorithm on a Quantum Computer. *Nature*, 392(quant-ph/9805069):344–346, 1998.

[Joz97] Richard Jozsa. Quantum Algorithms and the Fourier Transform. In *Proceedings of the Royal Society, London, Series A*, 1997.

[Llo00] Seth Lloyd. Ultimate physical limits to computation. *Nature*, 406:1047–1054, 2000.

[Mat06] Matthias Eck, Pawel Wocjan, & Robert M. Zeier. QuaSi - Quantum Circuit Simulator. `http://iaks-www.ira.uka.de/QIV/QuaSi/aboutquasi.html`, 2006. Acessado em março de 2007.

[Mee05] Marco Van Meegen. Checkclipse Plugin. `http://www.mvmsoft.de/content/plugins/checkclipse/checkclipse.htm`, 2005. Acessado em março de 2007.

[Moo65] Gordon E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics*, 38(8):114–117, 1965.

[Mos71] Joel Moses. Algebraic simplification: a guide for the perplexed. *Commun. ACM*, 14(8):527–537, 1971.

[NC00] Michael Nielsen & Issac Chuang. *Quantum computation and quantum information*. Cambridge, 2000.

[Obj04] Object Mentor. JUnit.org. `http://www.junit.org/index.htm`, 2004. Acessado em março de 2007.

[PLCM04] Renato Portugal, Carlile Campos Lavor, Luiz Mariano Carvalho, & Nelson Maculan. *Uma Introdução à Computação Quântica*. Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC, 2004.

[Rad02] Jim Radford. JMath: A GNU Readline based frontend for Mathematica. `http://robotics.caltech.edu/~radford/jmath/`, 2002. Acessado em março de 2007.

[RBFR06] Susan H. Rodger, Bart Bressler, Thomas Finley, & Stephen Reading. Turning automata theory into a hands-on course. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 379–383. ACM Press, 2006.

[Rou99] Marc R. Roussel. Redesigning the quantum mechanics curriculum to incorporate problem solving using a computer algebra system. *Journal of Chemical Education*, 76(10):1373–1377, 1999.

[Sav06] Nick Savoiu. Deus Ex Machina. `www1.ics.uci.edu/~savoiu/dem/`, 2006. Acessado em março de 2007.

[Sch00] Felix Schurmann. Interactive quantum computation. Dissertação de mestrado, University of New York, 2000.

[Sen06] Senko Corporation. World's First Universal - Quantum Computer Simulator. `www.senko-corp.co.jp/qcs/index.html`, 2006. Acessado em março de 2007.

[Sho94] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Sim94] Daniel R. Simon. On the power of quantum computation. In *Symposium on Foundations of Computer Science*, pages 116–123, 1994.

[Sun06a] Sun Microsystems. Java Technology. `http://java.sun.com/`, 2006. Acessado em março de 2007.

[Sun06b] Sun Microsystems. Javadoc Tool. `http://java.sun.com/j2se/javadoc/index.jsp`, 2006. Acessado em março de 2007.

[Wal99] Julia Wallace. A brief history of quantum computation. http://www.if.ufrgs.br/ jgallas/QUBITS/CURSO/brief_history.html, 1999.

[Wes99] Michael Wester. *A Critique of the Mathematical Abilities of CA Systems*, cap. 3. John Wiley & Sons, 1999.

[Wik07] Wikipédia. LaTeX. `http://pt.wikipedia.org/wiki/LaTeX`, 2007. Acessado em março de 2007.

[Wol05] Wolfram Research. Wolfram Research, Inc. `http://www.wolfram.com/`, 2005. Acessado em março de 2007.

[YYP01] Cecile Yehezkel, William Yurcik, & Murray Pearson. Teaching Computer Architecture with a Computer-Aided Learning Environment: State-of-the-Art Simulators. In *International Conference on Simulation and Multimedia in Engineering Education (ICSEE 2001)*, 2001.

Apêndice A

A notação de Dirac

Criada pelo inglês por Paul A. M. Dirac, a notação *braket* ou notação de Dirac, se tornou padrão na representação de estados quânticos. Esta é um meio útil para facilitar e agilizar a execução de operações no contexto da computação quântica.

Na notação de Dirac um vetor $\vec{v} = a_0\vec{v_0} + ... + a_n\vec{v_n}$, onde $\vec{v_0}$, ..., $\vec{v_n}$ formam uma base de um espaço V_n, é representado como $|v\rangle = a_0 |0\rangle + ... + a_n |N\rangle$, onde $|0\rangle$, ..., $|N\rangle$ formam uma base de um espaço V_n. A representação $|v\rangle$ é lida como '*ket* vê'. Com isso, utilizando notação matricial, tem-se:

$$|v\rangle = a_0 |0\rangle + ... + a_n |N\rangle = a_0 \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} + \cdots + a_n \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} \tag{A.1}$$

O vetor *hermitiano conjugado* (*adjunto*) de $|v\rangle$, representado por $\langle v|$, chamado de '*bra* vê', corresponde ao vetor:

$$\langle v| = a_0^* \begin{bmatrix} 1 & ... & 0 \end{bmatrix} + ... + a_n^* \begin{bmatrix} 0 & ... & 1 \end{bmatrix} = \begin{bmatrix} a_0^* & ... & a_n^* \end{bmatrix} \tag{A.2}$$

onde, $a_0, ..., a_n \in \mathbb{C}$, a_i^*, com $0 \leq i \leq n$, representa o conjugado de a_i. Também se pode representar o vetor adjunto de $|v\rangle$ como $|v\rangle^\dagger$ (lê-se † como dáguer).

O produto interno é representado na notação de Dirac como produto entre *bra* e *ket*, correspondendo ao complexo:

$$\langle u|v\rangle = \begin{bmatrix} u_0^* & ... & u_n^* \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} = u_0^* v_0 + ... + u_n^* v_n \tag{A.3}$$

O produto externo é representado na notação de Dirac como produto entre *ket* e *bra*, correspondendo a matriz:

$$|v\rangle \langle u| = \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} \begin{bmatrix} u_0^* & ... & u_n^* \end{bmatrix} = \begin{bmatrix} v_0 u_0^* & ... & v_0 u_n^* \\ \vdots & ... & \vdots \\ v_n u_0^* & ... & v_n u_n^* \end{bmatrix} \tag{A.4}$$

Pode-se interpretar a representação matricial sob a notação de Dirac como, *ket* correspondendo as linhas e *bra* correspondendo as colunas. Por exemplo, seja M uma matriz, esta é representada utilizando *braket* como a seguir:

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = a\,|0\rangle \langle 0| + b\,|0\rangle \langle 1| + c\,|1\rangle \langle 0| + d\,|1\rangle \langle 1| \tag{A.5}$$

Apêndice B

Lista de comandos do *CAS*

Todos os comandos suportados pelo *CAS* podem ser executados através dos botões personalizáveis ou utilizando o *prompt*. Os comandos disponíveis são sucintamente descritos abaixo:

- *ApplyAllGates* - aplica todos os *gates* existentes até a última coluna simulada;
- *ApplyGates* - aplica o primeiro *gate* existente na expressão;
- *ApplyGatesToCursor* - aplica todos os *gates* existentes até a coluna selecionada;
- *ApplyTensorProduct* - aplica o produto tensorial;
- *circuit.simulation.normal* - executa uma simulação normal;
- *circuit.simulation.reset* - reinicia uma simulação;
- *circuit.simulation.step-by-step* - executa um passo de simulação;
- *circuit.simulation.step-by-step-backward* - retorna um passo de simulação;
- *circuit.simulation.to-cursor* - executa uma simulação até a coluna selecionada;
- *conf.base(2)* - altera a base de representação para base binária;
- *conf.base(10)* - altera a base de representação para base decimal;
- *conf.gate.notation(1)* - representa os *gates* através de seus nomes;
- *conf.gate.notation(2)* - representa os *gates* através da notação de Dirac;
- *conf.gate.representation(1)* - exibe os *gates* antes dos *qubits* onde estes devem ser aplicados;
- *conf.gate.representation(2)* - exibe os *gates* no início da expressão;
- *conf.gate.showQubitIndex* - exibe/ignora o uso de índices identificadores nos *gates*;
- *conf.gate.showTensorial* - exibe/ignora o uso do símbolo $\otimes$ entre os *gates*;
- *conf.qubit.representation(1)* - altera a representação dos *qubits* para forma $|0\rangle ... |1\rangle$;

• *conf.qubit.representation(2)* - altera a representação dos *qubits* para forma $|0...1\rangle$;

• *conf.qubit.showQubitIndex* - exibe/ignora o uso de índices identificadores nos *qubits*;

• *conf.qubit.showTensorial* - exibe/ignora o uso do símbolo $\otimes$ entre os *qubits*;

• *Contraction* - altera a exibição de uma expressão para sua forma contraída, utilizando somatórios;

• *Delete* - exclui a expressão selecionada;

• *Evaluate* - avalia determinadas constantes (e.g. $cos(\frac{\pi}{2}) = 0, sen(\frac{\pi}{2}) = 1, ...$)

• *Expansion* - altera a exibição de uma expressão para sua forma expandida;

• *FactorConstants* - fatora/multiplica as constantes das amplitudes;

• *Input* - insere uma *string* em sintaxe LaTeX;

• *ShowIdentityGate* - exibe/ignora a representação de *gates* identidade;

• *Simplify* - solicita a execução de uma simplificação.

Apêndice C

Análise de desempenho (*Benchmark*)

A tabela seguinte descreve os circuitos e os tempos de execução obtidos para os respectivos circuitos com a alocação de 64 MB e 512MB.

Tabela C.1: Análise de desempenho com 64MB alocados

N^o de qubits	N^o de colunas	Portas	Tempo de resposta (t)
19	3	–	"imediato"
20	3	–	sem resposta após 5 min.
19	10	–	"imediato"
20	10	–	sem resposta após 5 min.
17	17	$X^{\otimes 17}$	"imediato"
18	18	$X^{\otimes 18}$	$t < 5$ segs.
19	19	$X^{\otimes 19}$	$t < 10$ segs.
20	20	$X^{\otimes 20}$	sem resposta após 5 min.
16	32	$X^{\otimes 16}Y^{\otimes 16}$	"imediato"
17	34	$X^{\otimes 17}Y^{\otimes 17}$	$t < 5$ segs.
18	36	$X^{\otimes 18}Y^{\otimes 18}$	$t < 10$ segs.
19	38	$X^{\otimes 19}Y^{\otimes 19}$	$t < 25$ segs.
20	40	$X^{\otimes 20}Y^{\otimes 20}$	sem resposta após 5 min.
15	45	$X^{\otimes 15}Y^{\otimes 15}Z^{\otimes 15}$	"imediato"
Continua na próxima página			

Tabela C.1 – continuação da página anterior

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
16	48	$X^{\otimes 16}Y^{\otimes 16}Z^{\otimes 16}$	$t < 5$ segs.
17	51	$X^{\otimes 17}Y^{\otimes 17}Z^{\otimes 17}$	$t < 10$ segs.
18	54	$X^{\otimes 18}Y^{\otimes 18}Z^{\otimes 18}$	$t < 15$ segs.
19	57	$X^{\otimes 19}Y^{\otimes 19}Z^{\otimes 19}$	$t < 35$ segs.
20	60	$X^{\otimes 20}Y^{\otimes 20}Z^{\otimes 20}$	sem resposta após 5 min.
9	8	$CNOT^{\otimes 8}$	“imediato”
10	9	$CNOT^{\otimes 9}$	$t < 5$ segs.
11	10	$CNOT^{\otimes 10}$	sem resposta após 5 min.
9	8	$Toffoli^{\otimes 7}$	“imediato”
10	9	$Toffoli^{\otimes 8}$	$t < 5$ segs.
11	10	$Toffoli^{\otimes 9}$	sem resposta após 5 min.
14	28	$H^{\otimes 14}X^{\otimes 14}$	“imediato”
15	30	$H^{\otimes 15}X^{\otimes 15}$	$t < 5$ segs.
16	32	$H^{\otimes 16}X^{\otimes 16}$	$t < 10$ segs.
17	34	$H^{\otimes 17}X^{\otimes 17}$	$t < 20$ segs.
18	36	$H^{\otimes 18}X^{\otimes 18}$	$t < 35$ segs.
19	38	$H^{\otimes 19}X^{\otimes 19}$	$t < 1$ min.
20	40	$H^{\otimes 20}X^{\otimes 20}$	sem resposta após 5 min.
13	39	$H^{\otimes 13}X^{\otimes 13}Y^{\otimes 13}$	“imediato”
14	42	$H^{\otimes 14}X^{\otimes 14}Y^{\otimes 14}$	$t < 5$ segs.
15	45	$H^{\otimes 15}X^{\otimes 15}Y^{\otimes 15}$	$t < 10$ segs.
16	48	$H^{\otimes 16}X^{\otimes 16}Y^{\otimes 16}$	$t < 15$ segs.
17	51	$H^{\otimes 17}X^{\otimes 17}Y^{\otimes 17}$	$t < 20$ segs.
18	54	$H^{\otimes 18}X^{\otimes 18}Y^{\otimes 18}$	$t < 40$ segs.
19	57	$H^{\otimes 19}X^{\otimes 19}Y^{\otimes 19}$	$t < 1$ min. 20 segs.
20	60	$H^{\otimes 20}X^{\otimes 20}Y^{\otimes 20}$	sem resposta após 5 min.
13	52	$H^{\otimes 13}X^{\otimes 13}Y^{\otimes 13}Z^{\otimes 13}$	“imediato”
Continua na próxima página			

Tabela C.1 – continuação da página anterior

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
14	56	$H^{\otimes 14}X^{\otimes 14}Y^{\otimes 14}Z^{\otimes 14}$	$t < 5$ segs.
15	60	$H^{\otimes 15}X^{\otimes 15}Y^{\otimes 15}Z^{\otimes 15}$	$t < 10$ segs.
16	64	$H^{\otimes 16}X^{\otimes 16}Y^{\otimes 16}Z^{\otimes 16}$	$t < 15$ segs.
17	68	$H^{\otimes 17}X^{\otimes 17}Y^{\otimes 17}Z^{\otimes 17}$	$t < 20$ segs.
18	72	$H^{\otimes 18}X^{\otimes 18}Y^{\otimes 18}Z^{\otimes 18}$	$t < 40$ segs.
19	76	$H^{\otimes 19}X^{\otimes 19}Y^{\otimes 19}Z^{\otimes 19}$	$t < 1$ min. 20 segs.
20	80	$H^{\otimes 20}X^{\otimes 20}Y^{\otimes 20}Z^{\otimes 20}$	sem resposta após 5 min.
9	17	$H^{\otimes 9}CNOT^{\otimes 8}$	“imediato”
10	18	$H^{\otimes 10}CNOT^{\otimes 9}$	$t < 5$ segs.
11	19	$H^{\otimes 11}CNOT^{\otimes 10}$	sem resposta após 5 min.
9	17	$H^{\otimes 9}Toffoli^{\otimes 7}$	“imediato”
10	18	$H^{\otimes 10}Toffoli^{\otimes 8}$	$t < 5$ segs.
11	19	$H^{\otimes 11}Toffoli^{\otimes 9}$	sem resposta após 5 min.

Tabela C.2: Análise de desempenho com 512MB alocados

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
20	3	–	“imediato”
21	3	–	$t < 5$.
22	3	–	$t < 10$.
23	3	–	sem resposta após 5 min.
20	10	–	“imediato”
21	10	–	$t < 5$.
22	10	–	$t < 10$.
23	10	–	sem resposta após 5 min.
17	17	$X^{\otimes 17}$	“imediato”
18	18	$X^{\otimes 18}$	$t < 5$ segs.

Continua na próxima página

Tabela C.2 – continuação da página anterior

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
19	19	$X^{\otimes 19}$	$t < 10$ segs.
20	20	$X^{\otimes 20}$	$t < 15$ segs.
21	21	$X^{\otimes 21}$	$t < 35$ segs.
22	22	$X^{\otimes 22}$	$t <$ 1min. 10 segs.
23	23	$X^{\otimes 23}$	sem resposta após 5 min.
15	30	$X^{\otimes 15}Y^{\otimes 15}$	"imediato"
16	32	$X^{\otimes 16}Y^{\otimes 16}$	$t < 5$ segs.
17	34	$X^{\otimes 17}Y^{\otimes 17}$	$t < 10$ segs.
18	36	$X^{\otimes 18}Y^{\otimes 18}$	$t < 15$ segs.
19	38	$X^{\otimes 19}Y^{\otimes 19}$	$t < 20$ segs.
20	40	$X^{\otimes 20}Y^{\otimes 20}$	$t < 50$ segs.
21	42	$X^{\otimes 21}Y^{\otimes 21}$	$t <$ 2min.
22	44	$X^{\otimes 22}Y^{\otimes 22}$	$t <$ 4min. 17segs.
23	46	$X^{\otimes 22}Y^{\otimes 22}$	sem resposta após 5 min.
15	45	$X^{\otimes 15}Y^{\otimes 15}Z^{\otimes 15}$	"imediato"
16	48	$X^{\otimes 16}Y^{\otimes 16}Z^{\otimes 16}$	$t < 5$ segs.
17	51	$X^{\otimes 17}Y^{\otimes 17}Z^{\otimes 17}$	$t < 10$ segs.
18	54	$X^{\otimes 18}Y^{\otimes 18}Z^{\otimes 18}$	$t < 20$ segs.
19	57	$X^{\otimes 19}Y^{\otimes 19}Z^{\otimes 19}$	$t < 45$ segs.
20	60	$X^{\otimes 20}Y^{\otimes 20}Z^{\otimes 20}$	$t <$ 1min 35segs.
21	63	$X^{\otimes 21}Y^{\otimes 21}Z^{\otimes 21}$	$t <$ 3min 10segs.
22	66	$X^{\otimes 22}Y^{\otimes 22}Z^{\otimes 22}$	sem resposta após 5 min.
10	9	$CNOT^{\otimes 9}$	"imediato"
11	10	$CNOT^{\otimes 10}$	$t < 5$ segs.
12	11	$CNOT^{\otimes 10}$	sem resposta após 5 min.
10	8	$Toffoli^{\otimes 8}$	"imediato"
11	9	$Toffoli^{\otimes 9}$	$t < 10$ segs.

Continua na próxima página

Tabela C.2 – continuação da página anterior

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
12	10	$Toffoli^{\otimes 10}$	sem resposta após 5 min.
14	28	$H^{\otimes 14}X^{\otimes 14}$	"imediato"
15	30	$H^{\otimes 15}X^{\otimes 15}$	$t < 5$ segs.
16	32	$H^{\otimes 16}X^{\otimes 16}$	$t < 10$ segs.
17	34	$H^{\otimes 17}X^{\otimes 17}$	$t < 20$ segs.
18	36	$H^{\otimes 18}X^{\otimes 18}$	$t < 35$ segs.
19	38	$H^{\otimes 19}X^{\otimes 19}$	$t < 1$ min.
20	40	$H^{\otimes 20}X^{\otimes 20}$	$t < 2$ min.
21	42	$H^{\otimes 21}X^{\otimes 21}$	$t <$ 3min. 50segs.
22	44	$H^{\otimes 22}X^{\otimes 22}$	sem resposta após 5 min.
13	39	$H^{\otimes 13}X^{\otimes 13}Y^{\otimes 13}$	"imediato"
14	42	$H^{\otimes 14}X^{\otimes 14}Y^{\otimes 14}$	$t < 5$ segs.
15	45	$H^{\otimes 15}X^{\otimes 15}Y^{\otimes 15}$	$t < 10$ segs.
16	48	$H^{\otimes 16}X^{\otimes 16}Y^{\otimes 16}$	$t < 15$ segs.
17	51	$H^{\otimes 17}X^{\otimes 17}Y^{\otimes 17}$	$t < 20$ segs.
18	54	$H^{\otimes 18}X^{\otimes 18}Y^{\otimes 18}$	$t < 40$ segs.
19	57	$H^{\otimes 19}X^{\otimes 19}Y^{\otimes 19}$	$t < 1$ min. 20 segs.
20	60	$H^{\otimes 20}X^{\otimes 20}Y^{\otimes 20}$	$t < 2$ min. 30 segs.
21	63	$H^{\otimes 21}X^{\otimes 21}Y^{\otimes 21}$	$t < 5$ min.
22	66	$H^{\otimes 22}X^{\otimes 22}Y^{\otimes 22}$	sem resposta após 5 min.
14	56	$H^{\otimes 14}X^{\otimes 14}Y^{\otimes 14}Z^{\otimes 14}$	"imediato"
15	60	$H^{\otimes 15}X^{\otimes 15}Y^{\otimes 15}Z^{\otimes 15}$	$t < 5$ segs.
16	64	$H^{\otimes 16}X^{\otimes 16}Y^{\otimes 16}Z^{\otimes 16}$	$t < 10$ segs.
17	68	$H^{\otimes 17}X^{\otimes 17}Y^{\otimes 17}Z^{\otimes 17}$	$t < 25$ segs.
18	72	$H^{\otimes 18}X^{\otimes 18}Y^{\otimes 18}Z^{\otimes 18}$	$t < 45$ segs.
19	76	$H^{\otimes 19}X^{\otimes 19}Y^{\otimes 19}Z^{\otimes 19}$	$t < 1$ min. 45 segs.
20	80	$H^{\otimes 20}X^{\otimes 20}Y^{\otimes 20}Z^{\otimes 20}$	$t < 3$ min. 15 segs.
Continua na próxima página			

Tabela C.2 – continuação da página anterior

Nº de qubits	Nº de colunas	Portas	Tempo de resposta (t)
21	84	$H^{\otimes 21}X^{\otimes 21}Y^{\otimes 21}Z^{\otimes 21}$	sem resposta após 5 min.
10	19	$H^{\otimes 10}CNOT^{\otimes 9}$	"imediato"
11	21	$H^{\otimes 11}CNOT^{\otimes 10}$	$t < 10$ segs.
12	23	$H^{\otimes 12}CNOT^{\otimes 11}$	sem resposta após 5 min.
10	19	$H^{\otimes 10}Toffoli^{\otimes 8}$	"imediato"
11	21	$H^{\otimes 11}Toffoli^{\otimes 9}$	$t < 10$ segs.
12	23	$H^{\otimes 12}Toffoli^{\otimes 10}$	sem resposta após 5 min.