Modelagem de Computação Quântica em Haskell

Amr Sabry *
Departamento de Ciência da Computação, Universidade de Indiana
sabry@cs.indiana.edu

RESUMO

O artigo desenvolve um modelo de computação quântica a partir da perspectiva da programação funcional. O modelo explica as ideias fundamentais da computação quântica a um nível de abstração que é familiar aos programadores funcionais. A O modelo também ilustra algumas das dificuldades inerentes à interpretando a mecânica quântica e destaca as diferenças entre a computação quântica e os modelos de computação tradicionais (funcionais ou não).

Categorias e Descritores de Assuntos

C.0 [Organizações de Sistemas Computacionais]: Geral; D.1 [Técnicas de Programação]: Programação Aplicativa (Funcional); F.0 [Teoria da Computação]: Geral

Termos Gerais

Algoritmos, Linguagens, Teoria

Palavras-chave

Haskell, Computação Quântica, Qubit, Emaranhamento, Valor Virtual, Adaptador

1. INTRODUÇÃO

A computação quântica evoca fortes conexões com a computação pura programação funcional: inclui uma noção incorporada de paralelismo (embora esta noção seja qualitativamente diferente daquela encontrada na programação funcional) e é com base em fundamentos matemáticos (espaços vetoriais, matrizes álgebra, etc.) que pode ser modelada elegantemente em um funcional linguagem [15].

Portanto, é natural modelar a computação quântica dentro de uma linguagem funcional. Como primeira aproximação, A computação quântica pode ser vista como uma extensão da computação clássica computação probabilística: com base em uma mônada de computações probabilísticas, pode-se desenvolver um modelo elegante, mas rudimentar, de computação quântica funcional [21]. Isso O artigo tenta oferecer um modelo mais completo de quântica computação em Haskell com dois objetivos principais em mente:

1. **Explicar a computação quântica em um nível de abstração familiar à comunidade de linguagens de programação em vez do modelo usado pelos físicos.**
2. **Para elicitar as conexões entre a computação quântica e programação funcional e avaliar a adequação das abstrações funcionais ao domínio de computação quântica.**

O primeiro objetivo é alcançado em um grau razoável. O O principal desafio aqui é que qualquer modelo operacional de computação quântica deve de alguma forma se comprometer com uma interpretação da mecânica quântica, que tem sido, e ainda é, objeto de debate entre os físicos. Em particular, nosso modelo deve implementar algum mecanismo para o colapso da função de onda inerente à medição. Usamos efeitos colaterais globais, independentemente de terem ou não algo a ver com a "realidade física".

Quanto ao segundo objetivo, argumentamos que, ao contrário das investigações preliminares, as abstrações de programação funcional (como realizado em Haskell) não são tão adequados à computação quântica. As incompatibilidades são, no entanto, bastante instrutivas. Primeiro, elas explicam parte da essência da computação quântica. computação, pois difere da programação funcional. A maioria significativamente, ao contrário do caso dos programas funcionais, o raciocínio sobre sistemas quânticos não é composicional, o que argumentamos que requer novas abstrações. Em segundo lugar, as incompatibilidades também expõe algumas das limitações do Haskell quando aplicado para um domínio radicalmente novo.

O restante do artigo está organizado da seguinte forma: Seção 2 apresenta os ingredientes básicos da computação quântica: estruturas de dados quânticas. A Seção 3 amplia o modelo com operações ou funções sobre dados quânticos. A Seção 4 aborda a questão espinhosa da observação ou medição. Seção 5 propõe um padrão de projeto que lembra o padrão Facade [12] e intimamente relacionado às referências componíveis de Kagawa [14] para manipular convenientemente componentes de estruturas de dados emaranhadas. A Seção 6 ilustra o modelo resultante fornecendo vários exemplos. Finalmente, a Seção 7 discute trabalho relacionado e conclui.

Tentamos tornar o artigo o mais independente possível possível, mas naturalmente não podemos incluir uma introdução completa à mecânica quântica e seus fundamentos matemáticos, nem podemos fazer um levantamento completo do campo da mecânica quântica.

* **Apoiado pela National Science Foundation sob Subvenções n.º CCR-0196063 e CCR-0204389.**

Haskell'03, 25 de agosto de 2003, Uppsala, Suécia.

computação. Convidamos o leitor a consultar artigos introdutórios clássicos [29, 23] para obter informações básicas adicionais sobre os conceitos e operações apresentados neste artigo.

2. QUANTOS DADOS

Os blocos de construção da computação quântica são qubits ou bits quânticos. Após explicar os qubits como um tipo de dado semelhante ao tipo Bool clássico , generalizamos a construção para outros tipos de dados.

2.1 Tipos enumerados

Em Haskell, o tipo de dados booleano e os construtores são definidos da seguinte forma:

```
dados Bool = Falso | Verdadeiro
```

Um valor do tipo Bool pode ser False ou True, mas nunca ambos ao mesmo tempo. Em contraste, qubits ou booleanos quânticos, cujo tipo denotamos QV Bool, são valores da seguinte forma geral:

a |Falso + b |Verdadeiro

onde ÿ e ÿ são números complexos que representam amplitudes de probabilidade, cada construtor c é interpretado como um vetor unitário |c, e + é a adição vetorial. Tal valor booleano quântico é, de alguma forma, Falso e Verdadeiro ao mesmo tempo, e essa superposição pode ser explorada em computações quânticas. Por exemplo, a superposição poderia ser explorada para explorar dois caminhos alternativos de uma computação em paralelo.

O uso de números complexos para representar as amplitudes de probabilidade significa que as amplitudes de probabilidade têm uma fase e, portanto, podem se reforçar ou se anular mutuamente durante cálculos intermediários. Em última análise, porém, os únicos observáveis ainda são Falso e Verdadeiro. A probabilidade de observar Falso ou Verdadeiro é proporcional ao quadrado da magnitude de ÿ e ÿ , respectivamente.

Generalizando de booleanos para tipos arbitrários a e seus versões quânticas QV a, observamos o seguinte:

- Todos os construtores para o tipo a serão interpretados como vetores unitários a partir dos quais podemos construir valores quânticos. Por conveniência, a lista de vetores unitários é um operador sobrecarregado para cada tipo de interesse: class (Eq a, Ord a) ÿ Base a onde base :: [a]

  Precisamos ser capazes de distinguir os vetores unitários entre si, portanto, exigimos que o tipo a seja um membro da classe Eq. Também exigimos que os vetores unitários estejam associados a uma ordem arbitrária, mas fixa (ou seja, que o tipo a seja um membro da classe Ord) para usar a biblioteca FiniteMap abaixo. Aqui estão alguns exemplos simples:

```
Movimento de dados = Vertical | Rotação de
dados horizontal = CtrClockwise | Cor de dados no
sentido horário = Vermelho | Amarelo | Azul

instância Base Bool onde base =
    [Falso, Verdadeiro]
instância Base Mover onde
    base = [Vertical, Horizontal] instância
Base Rotação onde
    base = [CtrClockwise, Sentido horário]
instância Base Color onde
    base = [Vermelho, Amarelo, Azul]
```

- Dados os vetores unitários para o tipo a, os valores do tipo QV a são mapas que associam cada vetor unitário a uma amplitude de probabilidade. Em muitos artigos sobre computação quântica, a identidade dos vetores unitários e sua ordem são mantidas implícitas, e os valores quânticos são representados usando apenas uma sequência de amplitudes de probabilidade.
  Embora isso pareça conveniente, não escala bem. Nossa representação de valores quânticos consistirá, portanto, em um mapa explícito de cada vetor unitário para sua amplitude de probabilidade. Abstratamente falando, esse mapeamento poderia ser realizado usando uma função, mas um protótipo inicial dessa ideia introduziu uma enorme perda de desempenho a ponto de mesmo alguns dos exemplos mais simples não poderem ser simulados. O problema é que, a menos que as funções sejam memorizadas, cada chamada de função recalcula o mapeamento. Mesmo em um pequeno exemplo de computação quântica, as funções normalmente seriam aninhadas em vários níveis e a desaceleração exponencial é inaceitável. Em vez disso, realizamos o mapeamento usando a biblioteca FiniteMap (que assume que o tipo a é uma instância da classe Ord, conforme necessário):

```
tipo PA = Complex Double tipo QV a
= FiniteMap a PA
```

Por convenção, vetores unitários associados a uma amplitude de probabilidade zero frequentemente serão omitidos do mapa finito. A função pr retorna a amplitude de probabilidade associada a um determinado vetor unitário:

```
qv :: Base a ÿ [(a, PA)] ÿ QV a qv = listToFM

pr :: Base a ÿ FiniteMap a PA ÿ a ÿ PA pr fm k =
lookupWithDefaultFM fm 0 k
```

Aqui estão alguns exemplos simples:

```
qFalse, qTrue, qFT :: QV Bool qFalse =
unitFM Falso 1 qTrue = unitFM
Verdadeiro 1 qFT = qv [(Falso, 1/ ÿ 2), (Verdadeiro, 1/ ÿ 2)]

qUp :: QV Mover qUp
= unidadeFM Vertical 1
```

O valor qFalse é um valor booleano quântico que é sempre Falso; da mesma forma, o valor qTrue é um valor quântico que é sempre Verdadeiro. O valor qFT é "meio-Falso" e "meio-Verdadeiro". O valor qUp é um valor quântico que é sempre Vertical. O fator de normalização de 1/ ÿ2 em qFT não é computacionalmente significativo.
e muitas vezes serão omitidos.

2.2 Tipos Infinitos

Em princípio, é possível estender as ideias da seção anterior para infinitos tipos de dados, como os números naturais:

```
instância Base Inteiro onde base = [0..]
```

Um valor quântico “bom” do tipo QV Inteiro associaria uma amplitude de probabilidade diferente de zero a apenas alguns vetores unitários, ou se associasse amplitudes de probabilidade diferentes de zero a todos os vetores unitários, as amplitudes deveriam “desaparecer rápido o suficiente” como:

```
qi :: QV Inteiro qi =
qv [ (i, 1 / i) | i ÿ base, i = 0]
```

Mas, além de sermos capazes de expressar valores quânticos como qi, pouco podemos fazer com eles, pelo menos se quisermos manter a apresentação e o código razoavelmente simples. Como ficará claro nas Seções 3 e 4, precisaremos realizar operações estritas, como adição nas amplitudes de probabilidade associadas a todos os vetores unitários. Quando o número de vetores unitários é finito, isso pode ser feito com a primitiva Haskell +; quando o número de vetores unitários é infinito, como no caso de Integer, precisaríamos manipular séries convergentes e integrais em vez de adições e somas. Não lidamos com tipos de dados infinitos no restante deste relatório.

2.3 Pares

Dados dois valores quânticos do tipo QV a e QV b, podemos construir dois tipos de pares: um do tipo (QV a, QV b) e um do tipo QV (a, b). O primeiro tipo de par não tem nada de especial: valores quânticos são como qualquer outro valor, pois podem ser armazenados em estruturas de dados. O segundo tipo de par é uma instância de uma noção mais geral de valores emaranhados, que possui diversas propriedades novas sem contrapartida no caso clássico e, portanto, requer um estudo cuidadoso.

Primeiro, dada a base para pares:

```
instância (Base a, Base b) ÿ Base (a, b) onde base =
  [ (a, b) | a ÿ base, b ÿ base ]
```

podemos escrever alguns exemplos:

```
p1, p2, p3 :: QV (Bool, Bool)

p1 = qv [((False, False), 1), ((False, True), 1)]

p2 = qv [((Falso, Falso), 1), ((Verdadeiro, Verdadeiro), 1)]

p3 = qv [((False, False), 1),
         ((Falso, Verdadeiro), 1),
         ((Verdadeiro, Falso), 1),
         ((Verdadeiro, Verdadeiro), 1)]
```

O primeiro componente do par quântico p1 é sempre Falso e o segundo é Falso ou Verdadeiro com igual probabilidade.
Isso pode ser formalizado dizendo que o par é equivalente ao produto tensorial dos valores qFalse e qFT. O produto tensorial é geralmente definido da seguinte forma:

```
(&ÿ) :: (Base a, Base b) ÿ
        QV a ÿ QV b ÿ QV (a, b) qa &ÿ qb =
qv [ ((a, b), pr
  qa a ÿ pr qb b) | (a, b) ÿ base ]
```

O fato de os dois componentes do par p1 poderem ser separados em dois valores não é uma propriedade geral dos pares quânticos. De fato, os componentes do par p2 não podem ser separados. A razão intuitiva é simples: cada componente do par p2 pode ser Falso ou Verdadeiro com igual probabilidade, o que sugere que o par pode ser igual a qFT &ÿ qFT.
Mas é claro que este produto tensorial produz p3 , que é bastante diferente de p2. Os componentes de um par como p2 que não podem ser separados estão emaranhados.

A situação para pares se generaliza para outros tipos de dados estruturados que também podem ser emaranhados. Valores emaranhados são um aspecto fundamental da computação quântica e serão revisitados com mais detalhes em seções posteriores. Notamos imediatamente, no entanto, que o emaranhamento implica que o raciocínio sobre sistemas quânticos não é composicional:

> Um aspecto surpreendente e não intuitivo do espaço de estado de um sistema quântico de n partículas é que o estado do sistema nem sempre pode ser descrito em termos do estado de suas partes componentes. [23, p.308]

3. FUNÇÕES/OPERAÇÕES

Na situação clássica, a única operação unária não trivial ação em booleanos é a função não definida da seguinte forma:

```
não Falso = Verdadeiro
não Verdadeiro = Falso
```

A função correspondente em valores booleanos quânticos mapeia o valor geral (ÿ |Falso + ÿ |Verdadeiro) para (ÿ |Falso + ÿ |Verdadeiro). Ela pode ser definida da seguinte forma:

```
qnotf :: QV Bool ÿ QV Bool qnotf v =
qv [(Falso, pr v Verdadeiro), (Verdadeiro,
                    pr v Falso)]
```

É fácil calcular que qnotf qFalse é avaliado como qTrue e vice-versa. Naturalmente, qnotf também pode ser aplicado a valores mistos, como qFT.

Devido à estrutura mais rica dos booleanos quânticos, é possível definir muito mais funções além do simples qnotf .
A função hadamardf abaixo mapeia um valor geral da forma (ÿ |Falso + ÿ |Verdadeiro) para ((ÿ + ÿ) |Falso + (ÿ ÿ ÿ) |Verdadeiro). Esta operação pode ser definida da seguinte forma:

```
hadamardf :: QV Bool ÿ QV Bool hadamardf
v = deixe ÿ = pr v
   Falso ÿ = pr v
       Verdadeiro em qv
   [(Falso, ÿ + ÿ), (Verdadeiro, ÿ ÿ ÿ)]
```

Um cálculo simples mostra que hadamardf qFalse é avaliado como qFT.

Deve estar claro neste ponto que existe um padrão geral para todas as operações em dados quânticos. O valor de saída tem uma amplitude de probabilidade associada a cada um de seus vetores unitários; e cada uma dessas amplitudes pode depender das amplitudes de probabilidade de todos os vetores unitários do valor de entrada. Em outras palavras, uma operação em dados quânticos é completamente especificada por uma matriz que especifica como cada amplitude de probabilidade de entrada contribui para cada amplitude de probabilidade de saída. Representamos essa matriz por outra aplicação finita:

```
dados Qop ab = Qop (FiniteMap (a, b) PA)

qop :: (Base a, Base b) ÿ [((a, b), PA)] ÿ Qop ab qop = Qop . listar
para FM
```

Para aplicar uma operação a um valor quântico, multiplicamos a matriz pelo vetor que representa o valor:

```
qApp :: (Base a, Base b) ÿ
        Qop ab ÿ QV a ÿ QV b
```

```
qApp (Qop m) v =
  deixe bF b = soma [ pr m (a, b) ÿ pr va | a ÿ base] em qv
  [(b, bF b) | b ÿ base ]
```

Por exemplo, as operações qnotf e hadamardf mencionadas acima podem ser definidas usando as seguintes matrizes:

```
qnotop = qop [((Falso, Verdadeiro), 1),
              ((Verdadeiro, Falso),
1)] hadamardop = qop [((Falso, Falso), 1),
                      ((Falso, Verdadeiro), 1),
                      ((Verdadeiro, Falso), 1),
                      ((Verdadeiro, Verdadeiro), ÿ1)]
```

Como outro exemplo, o operador a seguir traduz os estados de polarização vertical/horizontal de um fóton para os estados de polarização horário/anti-horário [19]:

```
m2r :: Rotação de movimento
Qop m2r = qop [((Vertical, CtrClockwise), 1),
               ((Vertical, Sentido horário), 1),
               ((Horizontal, CtrSentido horário), 0 :+ ÿ1),
               ((Horizontal, Sentido horário), 0 :+ 1)]
```

A notação a :+ b é a sintaxe de Haskell para o número complexo a + ib.

3.1 Elevação

Para entender melhor a natureza das operações quânticas, discutiremos brevemente uma maneira de elevar funções clássicas a operações sobre valores quânticos. A ideia básica é simples: um vetor unitário de entrada contribui para um vetor unitário de saída se, e somente se, a função clássica relacionar os construtores correspondentes:

```
opLift :: (Base a, Base b) ÿ (a ÿ b) ÿ Qop ab opLift f = qop [((a,
fa), 1) | uma base ÿ]
```

No entanto, isso nem sempre faz sentido, pois todas as operações quânticas devem ser unitárias (ou seja, a operação é invertível e, quando vista como uma matriz, o inverso da operação é apenas a transposta conjugada da matriz). No caso especial em que f é uma função reversível, a construção acima funciona e produz o que é chamado de operador pseudoclássico. Por exemplo, a função not é reversível e a construção acima de fato produz qnotop .

Outras funções clássicas como e não são reversíveis: a partir de uma saída False, não é possível calcular as duas entradas da função e . No entanto, uma função não reversível como e pode ser trivialmente tornada reversível adicionando saídas adicionais que transferem as entradas:

```
reversível e ab = (a, b, a && b)
```

Em geral, qualquer computação clássica, por mais complexa que seja, pode ser reversível ao se memorizar uma quantidade suficiente de seus resultados intermediários. Bennett mostra como uma Máquina de Turing universal pode ser simulada por uma Máquina de Turing reversível [4]. A ideia também pode ser adaptada a máquinas abstratas como a máquina SECD [17] e otimizada além do requisito ingênuo de armazenar todos os valores intermediários [5].
A computação reversível também é um tópico interessante com seus próprios méritos [11, 1].

3.2 Produzindo Pares Emaranhados Operações

controladas são a forma mais comum de introduzir emaranhamento em sistemas quânticos. A forma mais simples tal operação é a operação não controlada (cnot); cnot recebe dois valores booleanos quânticos e:

- não faz nada se o primeiro valor (chamado valor de controle) for Falso, e
- nega o segundo valor (chamado valor alvo) ou-comprovado.

Isso parece bastante simples até lembrarmos que o qubit de controle pode ser simultaneamente Falso e Verdadeiro. Nesse caso, a operação executa ambas as ações simultaneamente e o par resultante de qubits é emaranhado. Por exemplo, considere a situação em que o qubit de controle é qFT e o qubit alvo é qFalse. Como o qubit de controle é Falso com uma probabilidade diferente de zero, uma possível saída da operação é o estado (Falso, Falso). Além disso, como o qubit de controle é Verdadeiro com uma probabilidade diferente de zero, uma possível saída da operação é o estado (Verdadeiro, Verdadeiro). Nenhuma outra saída é possível. Em outras palavras, aplicar a operação cnot a qFT e qFalse produz o par emaranhado p2 da Seção 2.3.

De forma mais geral, a operação realizada no segundo valor não precisa ser uma negação, e o valor de controle não precisa ser um booleano. Abstraímos dessas duas situações para definir uma operação controlada genérica que recebe dois argumentos: um operador quântico u, que pode ser aplicado ao qubit alvo, e uma função clássica enable, que decide, para cada valor de controle a, se ele deve habilitar a aplicação da operação u ao alvo:

```
cop :: (Base a, Base b) ÿ (a ÿ
       Bool ) ÿ Qop bb ÿ Qop (a, b) (a, b) cop enable (Qop
u) = qop ([ (((a, b), (a, b)),
  1) | (a, b) ÿ base, não
       (habilitar a)] ++ [ (((a, b1), (a, b2)), pr u
     (b1, b2)) | a ÿ base, habilitar a, b1 ÿ
      base, b2 ÿ base])
```

Se o valor de controle de entrada não estiver habilitado, o par de saída será idêntico ao par de entrada (primeiro grupo); caso contrário, se o valor de controle de entrada estiver habilitado e for idêntico ao valor de controle de saída, a contribuição será aquela dada pelo operador u. Em todos os outros casos, a probabilidade de saída é zero e, portanto, omitida, seguindo nossa convenção.

A operação cnot é facilmente obtida a partir da operação genérica.

```
cnot :: Qop (Bool, Bool) (Bool, Bool) cnot
= id policial qnotop
```

Outra operação controlada comum é a operação controlada-controlada-não, também chamada de operação Toffoli [23].
A operação toffoli é essencialmente idêntica à operação cnot, mas é controlada por um par de valores booleanos. Sua definição é quase idêntica à de cnot:

```
toffoli :: Qop ((Bool, Bool ), Bool ) ((Bool, Bool ), Bool ) toffoli
= cop (uncurry (&&)) qnotop
```

A operação Toffoli é significativa porque pode implementar todas as operações booleanas clássicas. Quando ambos os valores de controle são True, a operação nega o valor de destino. Se o valor de destino for False, a operação executa uma função lógica "and" dos valores de controle. Como pode implementar "and" e "not", a operação pode implementar qualquer função booleana.

4. MEDIÇÃO

Valores, sejam eles resultantes de uma computação clássica ou quântica, devem ser observados para comunicar os resultados ao mundo exterior. Em um modelo de programação clássico, o processo de observação de um valor simplesmente retorna o valor. No modelo quântico, o processo de observação é mais complexo.

4.1 Normalização

Até o momento, não impusemos nenhuma restrição às amplitudes de probabilidade associadas aos vetores unitários de um valor quântico. Isso é válido, visto que a magnitude dos vetores não tem relevância computacional. No entanto, é útil ter uma representação normalizada antes de discutir os detalhes da medição. A normalização consiste simplesmente em dividir cada amplitude de probabilidade pela norma do valor. (No código abaixo, usamos | | para nos referir à função (quadrado . magnitude).) ^{2}

```
normalizar :: Base a ÿ QV a ÿ QV a normalizar v =
(1 / norma v :+ 0) ÿ> v

norma :: Base a ÿ QV a ÿ Norma dupla v =
deixe probs = map | (eltsFM v) em ÿ soma probs
                  (ÿ>) :: Base a ÿ

PA ÿ QV a ÿ QV ac ÿ> v = mapFM (\ a ÿ c ÿ a) v
```

Por exemplo, a normalização de p1, p2 **e** p3 **produz:**

```
np1 = qv [((False, False), 1/ ÿ 2),
          ((False, True), 1/ ÿ 2)] np2
= qv [((False, False), 1/ ÿ 2),
          ((Verdadeiro, Verdadeiro),
1/ ÿ 2)] np3 = qv [((Falso, Falso), 1/2),
          ((Falso, Verdadeiro), 1/2),
          ((Verdadeiro, Falso), 1/2),
          ((Verdadeiro, Verdadeiro), 1/2)]
```

4.2 Observando Valores Simples

Seja q **um valor booleano quântico normalizado** (ÿ **|False+** ÿ **|Verdadeiro) onde** $|ÿ|^{2} + |b|^{2} = 1$. **Uma medição de q:**

- **retorna um resultado res que é Falso com probabilidade** $|ÿ|^{2}$ **ou Verdadeiro com probabilidade** $|ÿ|^{2}$**;**
- **como efeito colateral atualiza** q **para que todas as observações futuras retornar res.**

Assim, assim que o valor q **é observado, qualquer superposição de Falso e Verdadeiro que possa estar presente desaparece, e o valor se torna um Falso puro ou um Verdadeiro puro.**

4.3 Observação e Emaranhamento

Dado um par de QV do tipo (a, b), a mecânica quântica permite três medições: uma medição do estado de

o par em si (ambos os componentes são medidos ao mesmo tempo); ou uma medição na qual o componente esquerdo ou o componente direito (mas não ambos) são medidos.

Em certo sentido, é bastante estranho que se possa operar individualmente sobre um dos componentes de um par emaranhado, mesmo que esse componente não possa ser separado do outro. De fato, o processo de observação fornece outra maneira de entender o emaranhamento. Dois valores estão emaranhados se a observação de um afeta a medição do outro. Revisando nossos exemplos da Seção 2.3, concluímos que os componentes de np3 não estão emaranhados e que os componentes de np2 estão emaranhados. De fato:

- **Cada componente de np3 é Falso e Verdadeiro com igual probabilidade, portanto, observar o primeiro componente pode retornar Falso ou Verdadeiro. Se retornar Falso, o par é "atualizado" (a função de onda colapsa) para ser consistente com esta observação e se torna:**

  qv [((False, False), 1/ ÿ 2), ((False, True), 1/ ÿ 2)]

  Uma observação futura do segundo componente ainda tem a mesma probabilidade de ser Falsa ou Verdadeira.

- **Em contraste, embora cada componente de np2 seja Falso e Verdadeiro com igual probabilidade, os valores são correlacionados. A observação do primeiro componente pode retornar Falso ou Verdadeiro. Se retornar Falso, o par é atualizado para:**

  qv [((False, False), 1)]

  e qualquer observação futura do segundo componente deve agora retornar Falso.

4.4 O Paradoxo EPR

A mecânica quântica descreve os fenômenos de entanglemento e observação sem interpretação:

> **É importante notar que não há um mecanismo postulado nesta teoria para explicar como uma função de onda é enviada a um autoestado por um observável. Assim como a lógica matemática não precisa exigir causalidade por trás de uma implicação entre proposições, a lógica da mecânica quântica não exige uma causa específica por trás de uma observação.**
> **No entanto, o debate sobre a interpretação da teoria quântica levou frequentemente os seus participantes a afirmar que a causalidade foi demolida na física. [16, p.6]**

Se quisermos fornecer um modelo operacional de computação quântica, precisaríamos de alguma interpretação da mecânica quântica para explicar como o segundo componente de um par é afetado quando o primeiro componente é observado. Para compreender as dificuldades, é útil revisar o famoso paradoxo de Einstein, Podolsky e Rosen [9] e algumas das tentativas de resolvê-lo.

Einstein, Podolsky e Rosen [9] propuseram um experimento complexo que utiliza valores emaranhados de uma maneira que parece violar os princípios fundamentais da relatividade. A questão é a seguinte: quando um componente de um par de valores emaranhados é observado, como a informação sobre o valor observado flui para o outro componente, se é que existe algum fluxo de informação em primeiro lugar! Existem duas tentativas padrão para resolver o paradoxo:

1. **A primeira tentativa de explicação é que cada componente do par possui um estado local que determina seu valor observado. Antes da observação, o estado local está oculto e só pode ser descrito probabilisticamente. Assim que o componente é observado, o estado oculto é exposto. No caso do par np2 acima de**

O estado oculto local de cada componente pode ser Falso; os componentes podem então ser observados em qualquer ordem e, sem qualquer comunicação ou interação, ambas as observações serão iguais, como esperado. Se válida, essa ideia produziria um modelo computacional simples e completamente local para a computação quântica. Infelizmente, Bell formula essa ideia matematicamente e mostra que ela é incompatível com as previsões estatísticas da mecânica quântica [2]. Bell conclui que qualquer teoria baseada em variáveis ocultas e que seja consistente com as previsões estatísticas da mecânica quântica também deve incluir um mecanismo pelo qual a configuração de um dispositivo de medição possa influenciar a leitura de outro instrumento, por mais remoto que seja, e que o sinal entre eles deve se propagar instantaneamente. Isso viola a relatividade especial.

2. A outra tentativa de explicação está intimamente relacionada à anterior: o valor de cada componente é uma função do valor medido do outro componente. O componente medido primeiro comunica seu valor ao outro componente, que o atualiza. Mas, como Einstein, Podolsky e Rosen observaram, essa explicação também viola os princípios da relatividade especial.

   A noção de um componente ser medido "primeiro" não é bem definida, pois depende da velocidade do agente que observa a medição. Em outras palavras, é possível que um observador veja que o componente esquerdo foi medido primeiro, enquanto outro observador vê que o componente direito foi medido primeiro. Em resumo, a ideia de comunicar um valor do primeiro componente a ser medido para o segundo componente não pode ser compatível com ambos os observadores, embora os experimentos sejam invariantes sob mudança de observador.

Infelizmente, embora essas duas explicações sejam reconhecidamente equivocadas, não existem outras explicações amplamente aceitas. Existem, no entanto, diversas interpretações interessantes que devem ser investigadas com mais profundidade, pois forneceriam modelos operacionais interessantes para a computação quântica: em particular, duas interpretações interessantes são a interpretação de muitos mundos, na qual todas as observações possíveis são realizadas em universos paralelos [10], e a interpretação transacional, na qual a computação é descrita como o ponto fixo de um processo que ocorre tanto no tempo direto quanto no tempo reverso [6].

Para os nossos propósitos, adotamos o mecanismo operacional mais simples para observar componentes de estruturas de dados emaranhadas, de modo que o resultado da observação afete todos os outros Valores emaranhados: usamos um efeito colateral global para uma referência compartilhada. A comunicação entre os valores emaranhados ocorre implícita e instantaneamente por meio da atribuição à referência compartilhada. Embora isso possa não ser sensato de uma perspectiva física, parece razoável em um ambiente de programação de thread única. Na presença de múltiplas threads (que não consideramos), um problema que lembra o observado por Einstein, Podolsky e Rosen pode ocorrer na forma de condições de corrida se duas threads tentarem medir diferentes componentes do par simultaneamente. Resta saber se o uso de efeitos colaterais globais em nosso modelo pode fazer com que simulações de computação quântica forneçam resultados e efeitos que não correspondem às contrapartes físicas.

4.5 Referências a Valores Quânticos

Para modelar os efeitos colaterais implícitos no processo de observação, usaremos referências explícitas: valores quânticos só podem ser acessados por meio de uma célula de referência; a observação atualiza a célula de referência com o valor observado:

```
dados QR a = QR (IORef (QV a))

mkQR :: QV a ÿ IO (QR a) mkQR v = do r
ÿ newIORef v return (QR r )
```

A função mkQR é uma ação de E/S que, quando executada, aloca uma nova célula de referência e armazena o valor quântico fornecido nela.

Para observar um valor quântico acessível por meio de uma referência QR a, lemos a referência de conteúdo, observamos o valor e atualizamos a referência com o resultado da observação. Observar um valor requer as seguintes etapas. Primeiro, normalizamos o valor. Em seguida, calculamos a probabilidade associada a cada vetor unitário na base. Para cada vetor unitário, também calculamos uma probabilidade cumulativa, que é a soma de sua probabilidade e de todas as probabilidades dos vetores unitários anteriores na ordem (arbitrária, pois irrelevante) dada pela base. Como as probabilidades somam 1, escolhemos um número aleatório entre 0 e 1 e escolhemos o primeiro construtor com uma probabilidade cumulativa que exceda esse número aleatório:

```
observeR :: Base a ÿ QR a ÿ IO a observeR (QR ptr ) =
do v ÿ readIORef ptr res ÿ
  observeV v writeIORef ptr
      (unitFM res 1) return res

observeV :: Base a ÿ QV a ÿ IO a observeV v =

  deixe nv = normalizar v
        probs = map (| | . pr nv)2 base r ÿ
      getStdRandom (randomR (0.0, 1.0)) deixe cPsCs = zip
      (scanl1 (+) probs) base
        Apenas( , res) = encontrar (\(p, ) ÿ r < p) cPsCs
      retornar res
```

Por exemplo, cada avaliação do teste abaixo imprime três ocorrências de Falso ou três ocorrências de Verdadeiro: a primeira observação tem a mesma probabilidade de ser Falso ou Verdadeiro, mas, uma vez realizada, ela corrige os resultados das próximas duas observações:

```
teste = faça x ÿ mkQR qFT o1 ÿ
          observeR x o2 ÿ
          observeR x o3 ÿ
          observeR x print (o1,
          o2, o3)
```

A observação de um dos componentes de um par é um pouco mais complicada. Mostramos apenas o caso da observação do componente esquerdo do par; o outro caso é simétrico:

```
observeLeft :: (Base a, Base b) ÿ
                  QR (a, b) ÿ IO a
observeLeft (QR ptr ) =
```

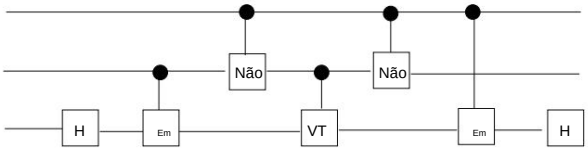
```
do v ÿ readIORef ptr
   deixe leftF a = soma [ |pr v (a, b)| 2 | b ÿ base] leftV = qv [(a,
      leftF a) | a ÿ base] aobs ÿ observeV leftV deixe
   nv = qv [((aobs, b), pr v (aobs,
   b)) | b ÿ base] writeIORef ptr (normalizar nv)
                  retornar aobs
```

Primeiro, construímos um valor quântico virtual leftV, que fornece a probabilidade associada a cada vetor unitário do componente esquerdo. Essa probabilidade é calculada somando todas as ocorrências desse vetor unitário no par. O valor virtual é observado e isso seleciona um dos vetores unitários.

O par é reconstituído apenas com os componentes que são consistentes com a observação, e o resultado é armazenado na célula de referência.

5. DUALIDADE ONDA/PARTÍCULA

Em princípio, cobrimos os fundamentos da computação quântica e podemos passar para alguns exemplos. Um exemplo elementar que consideramos é modelar esta implementação alternativa da operação de Toffoli:



O diagrama de circuito utiliza a notação padrão de fato para especificar cálculos quânticos. A convenção é que os valores fluem da esquerda para a direita em etapas correspondentes ao alinhamento das portas. Para o restante desta discussão, nos referimos aos três qubits relevantes como superior, médio e inferior:

1. **Na primeira etapa, a operação hadamard é aplicada ao qubit inferior.**

2. **Na segunda etapa, um vop controlado (definido abaixo) é aplicado ao par que consiste nos qubits do meio e da base:**

```
vop :: Qop Bool Bool vop =
qop [((Falso, Falso), 1),
                ((Verdadeiro, Verdadeiro), 0 :+ 1)]
```

3. **Na terceira etapa, a operação controlada cnot é aplicada ao par que consiste nos qubits superior e médio.**

4. **Na quarta etapa, uma operação controlada - vtop (a transposta adjunta ou conjugada de vop definida abaixo) é aplicada ao par que consiste nos qubits do meio e de baixo:**

```
vtop :: Saco Bool Bool vtop
= saco [((Falso, Falso), 1),
                ((Verdadeiro, Verdadeiro), 0 :+ ÿ1)]
```

5. **O quinto passo é idêntico ao terceiro passo.**

6. **Na sexta etapa, um vop controlado é aplicado ao par que consiste nos qubits superior e inferior. qubits.**

7. **Finalmente, na última etapa, a operação hadamard é aplicado ao qubit inferior.**

A implementação deste circuito bastante elementar é complicada pelo fato de que os três qubits (superior, médio e inferior) estão geralmente emaranhados. Não é possível manipular diretamente apenas o qubit inferior, como exigido pela primeira etapa, por exemplo. Pior ainda, o circuito exige que apliquemos operações a três pares distintos de qubits: (meio, inferior), (superior, médio) e (superior, inferior), que, novamente, por definição de emaranhamento, não podem ser isolados para se adequarem a cada operação.

Esta situação é a contrapartida de programação da dualidade onda/partícula: por um lado, os três valores emaranhados formam uma "onda" conectada; por outro lado, cada um deles é uma "partícula" independente que pode ser operada individualmente com o entendimento de que o resultado de tal operação afeta toda a onda.

A maneira ingênua de modelar computações como a acima é definir funções especializadas que operam em componentes de estruturas de dados, semelhantes à nossa função observeLeft da Seção 4.5. Isso rapidamente sai do controle e diversos modelos de computação quântica tentam fornecer um mecanismo geral para lidar com esse problema. Por exemplo, Selinger inclui operações que realizam permutações arbitrárias das variáveis [24], e QCL [22] inclui a noção de um registrador simbólico que pode se referir a qualquer coleção de qubits, mesmo que façam parte de estruturas emaranhadas. Em nosso caso, propomos uma ideia relacionada de valores virtuais.

5.1 Valores Virtuais e Adaptadores

Um valor virtual é um valor que, embora possivelmente embutido profundamente em uma estrutura e entrelaçado com outros, pode ser operado individualmente. Um valor virtual é especificado fornecendo toda a estrutura de dados à qual pertence e um adaptador que especifica o mapeamento de toda a estrutura de dados para o valor em questão e vice-versa. Mais especificamente, temos:

```
Adaptador de dados lg =
  Adaptador { dec :: g ÿ l, cmp :: l ÿ g }

dados Virt a na u = Virt (QR u) (adaptador (a, na) u)
```

O tipo (Virt a na u) define um valor virtual do tipo a que está entrelaçado com valores do tipo na. O tipo u é o tipo de toda a estrutura de dados que contém a e na. O adaptador mapeia para frente e para trás entre o tipo u e sua decomposição. Os valores virtuais estão relacionados a referências componíveis [14], que fornecem acesso a um campo ou subestrutura em relação a uma tupla ou registro maior usado como um

estado.

Por exemplo, em uma estrutura de dados do tipo

```
QV (((a, b, c), (d, e)), (f , g))
```

Existem várias maneiras de isolar um valor quântico do tipo QV(d, g), dependendo de como se decide agrupar os outros valores, com d e g emaranhados. Duas maneiras possíveis são:

```
mkVirt1 :: QR (((a, b, c), (d, e)), (f , g)) ÿ
                Virt (d, g) (a,
                    b, c, e, f ) (((a,
                    b, c), (d, e)), (f , g)) mkVirt1 r
= Virt r a1 onde a1 =
```

```
    Adaptador { dec =\ (((a, b, c), (d, e)), (f , g)) ÿ ((d, g),
                        (a, b, c, e, f )), cmp =\
                ((d, g), (a, b, c, e, f )) ÿ (((a, b, c), (d,
                        e)), (f , g)) }

mkVirt2 :: QR (((a, b, c), (d, e)), (f , g)) ÿ
          Virt (d, g)
               ((a, b, c), e, f )
               (((a, b, c), (d, e)), (f , g))
mkVirt2 r = Virt r a2 onde
  a2 =
    Adaptador { dec =\ (((a, b, c), (d, e)), (f , g)) ÿ ((d, g),
                        ((a, b, c), e, f )), cmp =\
                ((d, g), ((a, b, c), e, f )) ÿ (((a, b, c), (d,
                        e)), (f , g)) }
```

O mecanismo de valores virtuais nos permite fingir que existe um par de tipos (d, g) na estrutura, mesmo que o tipo (d, g) não ocorra diretamente no tipo da estrutura e os componentes dos tipos d e g estejam profundamente aninhados. Isso lembra o padrão Facade [12], no qual uma estrutura profundamente aninhada recebe uma interface plana que dá acesso às suas referências internas.

5.2 Gerando Adaptadores A definição de adaptadores (pelo menos para estruturas de dados como tuplas) é tão regular que deveríamos ser capazes de automatizar sua geração apenas a partir das informações de tipo. Assumiremos, no restante deste artigo, que os seguintes adaptadores foram gerados. Forneceremos apenas as definições para os dois primeiros:

```
par de anúncios1 :: Adaptador (a1,
a2) (a1, a2) par de anúncios1 = Adaptador {dec = (a1,
                                a2) ÿ (a1, a2), cmp = (a1, a2) ÿ (a1, a2) }
ad_pair2 :: Adaptador (a2, a1) (a1, a2)
ad_pair2 = Adaptador { dec = (a1, a2) ÿ (a2, a1), cmp =
                                (a2, a1) ÿ (a1, a2) }
anúncio triplo23 . . .
anúncio triplo12 . . .
anúncio triplo13 . . .
```

5.3 Tudo é um Valor Virtual. Para fornecer um modelo uniforme, reformulamos todas as nossas operações em termos de valores virtuais. Primeiro, fornecemos uma maneira de converter referências individuais a valores quânticos em valores virtuais triviais e uma maneira de criar valores virtuais a partir de outros valores virtuais, compondo um novo adaptador:

```
virtFromR :: QR a ÿ Virt a () a virtFromR r =

  Virt r (Adaptador { dec = \ a ÿ (a, ()), cmp = \
                      (a, ()) ÿ a })

virtFromV :: Virt a na u ÿ Adaptador (a1, a2) a ÿ
                 Virtude a1 (a2, na) u
virtFromV
  (Virt r (Adaptador { dec = gdec, cmp = gcmp }))
  (Adaptador {dec = ldec, cmp = lcmp }) =
   Virt r
    (Adaptador {dec = \u ÿ deixe (a, na) = gdec u (a1, a2)
                        = ldec a in (a1, (a2,
                    na)), cmp = \ (a1,
                (a2, na)) ÿ gcmp (lcmp(a1,
                        a2), na)})
```

Uma operação sobre valores quânticos recebeu anteriormente o tipo Qop ab, denotando o fato de que ela mapeia valores quânticos do tipo QV a para valores quânticos do tipo QV b. Em vez de valores quânticos simples como antes, os valores de entrada e saída agora são virtuais, ou seja, são do tipo Virt a na ua e Virt b nb ub. A operação do tipo Qop ab ainda deve fazer sentido, já que os valores de entrada e saída são do tipo correto, exceto pelo fato de estarem entrelaçados em estruturas maiores.
No entanto, a aplicação não afeta esses valores entrelaçados circundantes, que devem, portanto, ter o mesmo tipo.
Portanto, a aplicação geral é definida da seguinte forma:

```
app :: (Base a, Base b, Base nab,
        Base ua, Base ub) ÿ
    Qop ab ÿ Virt a nab ua ÿ Virt b nab ub ÿ


Aplicativo IO () (Qop f)
    (Código QR)
        (Adaptador { dec = deca, cmp = cmpa }))
    (Virt (QR rb)
        (Adaptador { dec = decb, cmp = cmpb })) = deixe
  gf = qop [((ua,
              ub), pr f (a, b)) | ua ÿ base,
               ub ÿ base, deixe (a, na) =
               deca ua, deixe (b, nb) =
               decb ub, na == nb] em
               do fa ÿ
  readIORef ra deixe fb =
      normalize $ qApp gf fa writeIORef rb
      fb
```

O primeiro argumento é a operação a ser aplicada. Os dois argumentos seguintes são os valores virtuais de entrada e saída que compartilham os mesmos vizinhos emaranhados. A operação é promovida de algo agindo sobre a e b para algo agindo sobre toda a estrutura emaranhada da maneira esperada.

Em geral, os valores virtuais de entrada e saída podem ser diferentes. Por exemplo, dado um valor virtual

```
ip :: Virt Move Bool (Move, Bool )
```

e um valor virtual

```
op :: Virt Rotation Bool (Bool, Rotação)
```

podemos usar o aplicativo m2r ip op para traduzir de um estado de polarização de um fóton para outro, mesmo quando o fóton estiver emaranhado com algum qubit.

Em exemplos simples, é mais comum ter apenas uma referência global a um valor quântico do tipo QV u, que é atualizado repetidamente no local por operações sucessivas. Cada uma das operações sucessivas é do tipo Qop aa para algum tipo a que pode ser extraído de u por meio de um adaptador. Para essas aplicações, podemos usar a seguinte versão mais simples de app:

```
app1 :: (Basis a, Basis na, Basis ua) ÿ
    Qop aa ÿ Virt a na ua ÿ IO () app1 fv =
app fvv
```

Um valor virtual pode ser observado usando uma ideia que generaliza observeLeft da Seção 4.5 usando o adaptador para

decompor e compor o valor em vez do conhecimento embutido de que estamos manipulando o componente esquerdo de um par:

```
observeVV :: (Base a, Base na, Base u) ÿ
                    Virt a na u ÿ IO a
observeVV (Virt (QR r )
                  (Adaptador { dec = dec, cmp = cmp })) =
do v ÿ readIORef r
   deixe virtF a = soma [ | pr v (cmp(a, na))| na ÿ base ]  2  |
                                deixe virtV = qv
   [(a, virtF a) | a ÿ base] aobs ÿ observeV virtV deixe
   nv = qv [(u, pr v (cmp(aobs,
   na))) | u ÿ base, deixe (a, na) = dec u, a == aobs]
                    writeIORef r
                    (normalizar nv) retornar
                    aobs
```

6. EXEMPLOS O maquinário

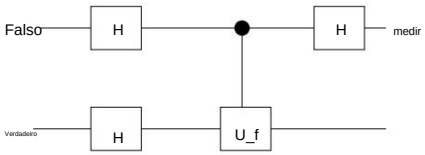
que desenvolvemos pode parecer bastante pesado, mas é bastante potente e torna a programação de diagramas de circuitos como o exemplo do Toffoli na Seção 5 bastante simples. O código completo (excluindo os adaptadores) é:

```
toffoli :: (Base em, Base em) ÿ
                Virt (Bool, Bool, Bool) na u ÿ IO () toffoli
vtriple = deixe b =
  virtFromV vtriple ad triple3
      mb = virtFromV vtriplo anúncio triplo23 tm
      = virtFromV vtriplo anúncio triplo12 tb =
      virtFromV vtriplo anúncio triplo13 cv = id
      do policial vop cvt
      = id do policial vtop
  em do app1 hadamardop b
        app1 cv mb
        app1 cnot tm
        app1 cvt mb
        app1 cnot tm
        app1 cv tb
        app1 hadamardop b
```

Dados quaisquer três valores booleanos quânticos (emaranhados ou não; parte de uma estrutura de dados maior ou não), começamos isolando as partes relevantes e, em seguida, simplesmente aplicamos as operações da maneira óbvia: uma linha para cada etapa do circuito. Usamos os mnemônicos mb para nos referirmos ao par de valores do meio e da base, tm para nos referirmos ao par de valores do topo e do meio, e assim por diante.

6.1 O Oráculo de Deutsch. Outro exemplo

interessante é o oráculo de Deutsch [7], que, dada uma função sobre booleanos, decide, com apenas uma invocação da função, se a função é balanceada (id ou não) ou constante (const True ou const False). É claro que a simulação de Haskell aplica a função tanto a True quanto a False, mas uma implementação quântica real aplicaria a função uma vez à superposição quântica. O exemplo não requer, de fato, o mecanismo de valores virtuais, mas utiliza o poder da operação controlada genérica da Seção 3.2:




```
deutsch :: (Bool ÿ Bool ) ÿ IO () deutsch f =
do inpr ÿ
mkQR (qFalse &ÿ qTrue) let both = virtFromR
    inpr top = virtFromV both ad
        pair1 bot = virtFromV both ad pair2 uf
        = cop f qnotop app1 hadamardop top
        app1 hadamardop bot
    app1 uf both app1
    hadamardop top topV ÿ
    observeVV top
    putStr (se topV então
    ”Balanceado” senão
    ”Constante”)
```

O oráculo funciona da seguinte maneira. O valor superior é transformado pela operação hadamard em |False + |True e o valor inferior é transformado em |False ÿ |True. Existem vários casos, dependendo de f:

- Se f for constante False: a linha de controle é sempre desabilitada e os valores superior e inferior permanecem inalterados. O último hadamard transforma o valor superior |False + |True em (|False + |True) + (|False ÿ |True), o que simplifica para |False se ignorarmos o fator de normalização como de costume.

- f é id: a linha de controle é uma superposição |Falso + |Verdadeiro e o valor inferior é deixado inalterado e negado de forma que fica entrelaçado com o valor superior. Mais precisamente, o par resultante de valores superior e inferior é:

  |(False, False) ÿ |(False, True)
  + |(Verdadeiro, Verdadeiro) ÿ |(Verdadeiro, Falso)

  que pode ser explicado da seguinte forma. Os dois primeiros componentes correspondem aos casos em que o valor superior é Falso; como f Falso também é Falso, a linha de controle é desabilitada e o valor inferior é |Falso ÿ |Verdadeiro.
  Os dois últimos casos correspondem aos casos em que o valor superior é Verdadeiro; como f Verdadeiro também é Verdadeiro, a linha de controle é habilitada e o valor inferior torna-se |Verdadeiro ÿ |Falso. Por fim, o valor superior é operado por hadamard, deixando o valor inferior intacto.
  Isso produz:

  |(False, False) + |(True, False)
  ÿ |(Falso, Verdadeiro) ÿ |(Verdadeiro, Verdadeiro)
  + |(Falso, Verdadeiro) ÿ |(Verdadeiro, Verdadeiro)
  ÿ |(False, False) + |(True, False)

  que simplifica para: |(Verdadeiro, Falso) ÿ |(Verdadeiro, Verdadeiro)
  Portanto, observar o valor superior (esquerdo) sempre retorna Verdadeiro.

- As situações em que f é constante True ou não são como as descritas acima. Caso f seja constante True, a linha de controle está sempre habilitada e o valor inferior é sempre negado e, portanto, não está emaranhado com o valor superior. Caso f não seja, os valores estão emaranhados e um resultado semelhante

A análise mostra que o valor superior (esquerdo) é avaliado como Verdadeiro. Portanto, em todos os casos, se o valor superior for observado como Falso, a função é constante, e se o valor superior for observado como Verdadeiro, a função é balanceada.

6.2 Somador Quântico

Um somador quântico de 1 bit pode ser definido usando portas Toffoli e portas não controladas [23]. Os principais destaques do código são:

```
somador :: QV Bool ÿ QV Bool ÿ QV Bool ÿ IO () somador inc xy =
let sum = qFalse
   outc = qFalse somador
        inputs = inc &ÿ x
        &ÿ y &ÿ sum &ÿ outc in do r ÿ mkQR vals let v = virtFromV
   (virtFromR r ) . . .

        . . .
        app1 toffoli vxyo
        app1 toffoli vixo
        app1 toffoli viyo
        app1 cnot vis
        app1 cnot vxs
        app1 cnot vys
        (sum, out carry) ÿ observeR vso print
        (sum, out carry)
```

No código, omitimos os adaptadores. Os valores virtuais denominados v com subscritos usam as seguintes convenções: i refere-se ao qubit de transporte, x e y referem-se aos dois qubits a serem somados, s refere-se ao qubit de soma e o refere-se ao qubit de transporte de saída. Portanto, vixo é o valor virtual referente aos três qubits: transporte de entrada, primeira entrada e transporte de saída.
O somador pode ser chamado com qFalse qTrue qTrue, caso em que ele age como um somador clássico e produz (False, True), mas também pode ser chamado com qFT qFT qFT .

7. CONCLUSÕES

Apresentamos um modelo de computação quântica embarcado em Haskell. Esperamos que o modelo forneça boas intuições sobre computação quântica para programadores. Utilizamos o modelo para escrever diversos outros algoritmos, incluindo o algoritmo de fatoração de Shor [26]. Para exemplos mais complexos do que os apresentados aqui, é útil ter um tipo de inteiros módulo n para permitir a parametrização conveniente de algoritmos em relação ao tamanho da entrada. Isso pode ser codificado usando classes de tipos [13, 20], mas de forma complexa, e talvez pudesse se beneficiar de extensões de metaprogramação para Haskell [25].
Além disso, embora acreditemos que a ideia de valores virtuais seja a correta, talvez sua execução atual deixe muito espaço para melhorias.

Nosso modelo evoca uma diferença fundamental entre linguagens de programação clássicas e linguagens de programação quântica. Na teoria das linguagens de programação clássicas, as expressões da linguagem podem ser agrupadas em construções de introdução e construções de eliminação para os conectivos de tipo da linguagem. Uma linguagem de programação quântica só pode ter construções de eliminação virtual porque, por definição, os elementos de uma estrutura de dados emaranhada não podem ser separados. Essa restrição leva a um estilo de programação incomum que, argumentamos, requer novas primitivas de programação.

Nosso modelo se distingue de outros trabalhos sobre computação quântica e programação funcional da seguinte forma. Ambos Skibinski [27] e Karczmarczuk [15] utilizaram Haskell extensivamente para modelar as estruturas matemáticas subjacentes à mecânica quântica, um foco diferente e complementar ao nosso. Skibinski [28] também implementou um simulador Haskell para computação quântica que opera no nível "físico" de abstração de qubits e portas, do qual tentamos abstrair usando tipos de dados e funções abstratas.

Também houve diversas propostas para “linguagens de programação quântica” que não se relacionam com a programação funcional [22, 8, 24]. Tanto a pGCL, uma linguagem imperativa que estende a linguagem de comando guardado de Dijkstra [8], quanto a QPL, uma linguagem funcionalmente tipada com dados quânticos [24], são bem desenvolvidas e semanticamente bem fundamentadas, podendo fornecer conexões interessantes com a programação funcional.

Agradecimentos Gostaríamos de agradecer aos revisores anônimos pelos comentários extensos e muito úteis.

8. REFERÊNCIAS

[1] HG Baker. NEVERSAL of fortune - a Termodinâmica da coleta de lixo. Em Y. Bekkers e J. Cohen, editores, Gerenciamento de Memória; Workshop Internacional IWMM 92; Anais, páginas 507–524, Berlim, Alemanha, 1992. Springer-Verlag.

[2] JS Bell. Sobre o paradoxo de Einstein-Podolsky-Rosen. Em [3], páginas 14–21. Cambridge University Press, 1987.

[3] JS Bell. O falável e o indizível na mecânica quântica. Cambridge University Press, 1987.

[4] CH Bennett. Reversibilidade lógica da computação. IBM Journal of Research and Development, 17(6):525–532, novembro de 1973.

[5] H. Buhrman, J. Tromp e P. Vit´anyi. Tempo e Limites espaciais para simulação reversível. Notas de Aula em Ciência da Computação, 2076:1017–1027, 2001.

[6] JG Cramer. A interpretação transacional da mecânica quântica. Física Moderna, 58:647–688, 1986.

[7] D. Deutsch. Teoria quântica, a teoria de Church-Turing princípio e o computador quântico universal. Proc. Roy. Soc. Londres, Ser. A, 400:97–117, 1985.

[8] EW Dijkstra. Uma Disciplina de Programação, capítulo 14. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[9] A. Einstein, B. Podolsky e N. Rosen. Pode A descrição quântico-mecânica da realidade física pode ser considerada completa? Phys. Rev., 47:777–780, 1935.

[10] H. Everett, III. Formulação de “estado relativo” de mecânica quântica. Revisões de Física Moderna, 29:454, 1957.

[11] MP Frank. Reversibilidade para computação eficiente. Tese de doutorado, MIT, 1999.

[12] E. Gamma, R. Helm, R. Johnson e J. Vlissides. Padrões de Projeto: Elementos de Software Reutilizável Orientado a Objetos. Série Computação Profissional. Addison-Wesley, 1995.

[13] R. Hinze. Haskell faz isso com classe. Slides de uma palestra dada na reunião de Haskell Genérico, maio de 2001.

[14] K. Kagawa. Estruturas de dados mutáveis e referências componíveis em uma linguagem funcional pura. Em Estado em

Linguagens de Programação (SIPL'95), páginas 79–94, janeiro de 1995.

[15] J. Karczmarczuk. Estrutura e interpretação da mecânica quântica — uma estrutura funcional. Em ACM SIGPLAN Haskell Workshop, 2003.

[16] LH Kauffman. Topologia Quântica e Computação Quântica, capítulo IV de [18]. Sociedade Matemática Americana, 2002.

[17] W. Kluge. Uma máquina SE(M)CD reversível. No 11º Workshop Internacional sobre Implementação de Linguagens Funcionais, Lochem, Holanda, 7 a 10 de setembro de 1999, número 1868 em Notas de Aula em Ciência da Computação, páginas 95–113. Springer-Verlag, setembro de 2000.

[18] SJ Lomonaco, Jr., editor. Computação Quântica: Um Grande Desafio Matemático para o Século XXI e o Milênio, volume 58 de Anais de Simpósios em Matemática Aplicada. Sociedade Americana de Matemática, março de 2002.

[19] SJ Lomonaco, Jr. Uma Pedra de Roseta para a Mecânica Quântica com uma Introdução à Computação Quântica, capítulo I de [18]. Sociedade Matemática Americana, 2002.

[20] C. McBride. Fingindo — simulando tipos dependentes em Haskell. Journal of Functional Programming, 12(4&5):375–392, julho de 2002.

[21] S.-C. Mu e R. Bird. Programação quântica funcional. Em Segundo Workshop Asiático sobre Linguagens e Sistemas de Programação, KAIST, Coreia, dezembro de 2001.

[22] B. Omer. Um formalismo procedural para computação quântica. Dissertação de mestrado, Departamento de Física Teórica, Universidade Técnica de Viena, 1998.

[23] E. Rieffel e W. Polak. Uma introdução à computação quântica para não físicos. ACM Computing Surveys, 32(3):300–335, set. 2000.

[24] P. Selinger. Rumo a uma linguagem de programação quântica. Não publicado, 2002.

[25] T. Sheard e S. Peyton-Jones. Modelo metaprogramação para Haskell. Em Proc. do workshop sobre Haskell, páginas 1–16. ACM, 2002.

[26] PW Shor. Algoritmos de tempo polinomial para fatoração de primos e logaritmos discretos em um computador quântico. SIAM Journal on Computing, 26(5):1484–1509, 1997.

[27] J. Skibinski. Coleção de módulos Haskell. Disponível em http://web.archive.org/web/20010415043244/ www.numeric-quest.com/haskell/index.html, **Inicializado em: 18/09/1998, última modificação em: 02/04/2001.**

[28] J. Skibinski. Simulador Haskell de computador quântico. Disponível em
http://web.archive.org/web/20010630025035/www. numeric-quest.com/haskell/QuantumComputer.html, **Inicializado em: 2001-05-02, última modificação em: 2001-05-05.**

[29] A. Steane. Computação quântica. Relatórios de progresso em Física, 61:117–173, 1998.