

Projeto e Implementação de uma Linguagem de Programação para Bioinformática

Felipe Fernandes Albrecht¹

¹ Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

albrecht@inf.furb.br

Resumo. A bioinformática é um campo científico formado pela genética molecular, estatística e ciências da computação. Uma das principais tarefas da bioinformática é a análise e estudo de seqüências genéticas e proteicas. Estas tarefas são feitas por meio de ferramentas computacionais construídas utilizando diversas linguagens, entre as quais C, C++, Java e Perl. Estas linguagens cumprem o propósito de construção de ferramentas para bioinformática, porém muitas vezes são demasiadamente complexas para os usuários que não são da área da ciência da computação. Para auxiliar a resolver este problema, é proposto neste artigo uma linguagem de programação de propósito específico, sendo ele, manipulação de seqüências genéticas e proteicas. A linguagem, chamada BioTongue, possui as seguintes características: facilidade de criar Tipos Abstratos de Dados e suas operações, possuir embutida na linguagem os principais tipos utilizados na bioinformática e uma sintaxe simples, clara e objetiva para tarefas de bioinformática. Este artigo apresenta a Linguagem BioTongue e o seu processo de desenvolvimento.

Abstract. Bioinformatics is a scientific field formed by molecular genetics, statistics and computer sciences. One of the main tasks of the bioinformatics is the analysis and study of genetic and proteinic sequences. These tasks are executed by constructed computational tools using diverse languages, between which C, C++, Java and Perl. These languages fulfill the intention of construction of tools for bioinformatics, however many times are too complex for the users who are not in the area of the computer science. To help to solve this problem, a programming language of specific intention, being it, manipulation of genetic and proteic sequences, is considered in this article. The language, called BioTongue, possesss the following characteristics: easiness to create Abstract Data Type and its operations, to possess inlaid in the language the main types used in the bioinformatics and a simple, clear and objective syntax for bioinformatics tasks. This article presents the BioTongue Language and its development process.

1. Introdução

A comparação e análise de seqüências genéticas e proteicas é parte de um campo científico chamado bioinformática. Este campo científico é baseado numa interação entre a estatística, biologia molecular e ciência da computação. A tarefa de comparação e análise de seqüências genéticas é feito com auxílio de ferramentas computacionais. Esta ferramentas são construídas utilizando diversas linguagens, entre elas ressalta C, C++, Java,

Perl e Python. Estas linguagens cumprem o seu propósito de construção de ferramentas, porém muitas vezes são demasiadamente complexas para os usuários que não são da área da computação. Isto pode ocorrer por utilizarem paradigmas complexos de programação, como a Orientação a Objetos, no caso de Java, ou por não possuir boa organização do fonte, como em Perl ou não abstrair detalhes como ponteiros, por exemplo em C e C++.

Para auxiliar no avanço da bioinformática e proporcionar que pesquisadores de outras áreas, especialmente da biologia, possam criar suas próprias ferramentas de pesquisa sem terem que recorrer a linguagens que não atendem as suas necessidades. Este artigo propõe uma linguagem de programação de propósito específico, sendo este, o trabalho com dados da genética molecular, como seqüências genéticas e proteicas.

Na próxima seção é apresentada a bioinformática, os motivos que levaram a criação de uma nova linguagem e as suas principais características são discutidos na seção 3. Na seção 4 a linguagem e seu funcionamento são apresentados e na seção 5 é exposto o método de desenvolvimento e implementação do compilador e máquina virtual. Por fim é dito sobre o estado atual da implementação do compilador e da máquina virtual da linguagem e aonde obtê-los e acompanhar seus desenvolvimentos.

2. Bioinformática

Segundo [Mount 2004], bioinformática foi definida como um campo interdisciplinar envolvendo biologia, ciências da computação, matemática e estatística para analisar dados de seqüências biológicas, conteúdo e organização do genoma e prever a função e estrutura de macromoléculas. A genética molecular é um campo da biologia que segundo [Srachan 2002] trata das inter-relações entre as macromoléculas de informação - o DNA(ácido desoxirribonucléico) e o RNA (ácido ribonucléico) - e de como elas são utilizadas para sintetizar polipetídeos, os componentes básicos de todas as proteínas. Em alguns vírus, o RNA é o material hereditário, mas, em todas as células, a informação genética está armazenada nas moléculas de DNA. A genética molecular possui um dogma central, que segundo [Durbin et al. 1998] é: a partir das seqüências de DNA são feitas as seqüências de RNA e estas são os moldes para as proteínas. Além disso, as seqüências de DNA e RNA são formados por quatro bases de ácidos nucleicos e as proteínas são formadas por um conjunto de vinte aminoácidos. Para o leitor que não está familiarizado com os estes conceitos ou que busca mais informações, é recomendada a leitura de [Lewin 2001].

Conforme dito na seção 1, a bioinformática utiliza a comparação e análise de seqüências genéticas. As comparações e análises são feitas por ferramentas computacionais que são desenvolvidas em diversas linguagens. O problema é que os pesquisadores da bioinformática necessitam criar suas próprias ferramentas, porém, as linguagens existentes não cumprem totalmente o seu dever. Isto ocorre porque as tarefas da bioinformática são baseadas em comparações de seqüências, ou seja, cadeias de base ou aminoácidos, e em operações matemáticas e as linguagens não proporcionam a abstração desejada para dados genéticos e suas operações, cabendo ao usuário arcar com esta complexidade.

3. Projeto da Linguagem BioTongue

Através de pesquisas nos principais *softwares* e bibliotecas de linguagens para bioinformática chegou-se a conclusão que as linguagens mais utilizadas nesta área são C, C++, Perl e Java. Nesta seção suas principais características serão analisadas. Outras linguagens de programação também são utilizadas na bioinformática, porém optou-se por estas quatro, porque são as de maior presença. O objetivo desta análise é formar uma lista de características presentes nas linguagens utilizadas e a partir dela formular os requisitos funcionais e não funcionais da Linguagem BioTongue.

3.1. Análise das linguagens atuais

A primeira linguagem analisada foi a Linguagem C [Kernighan ; Ritchie 1988]. A Linguagem C oferece como características fundamentais o seu baixo *overhead* e compilação e execução em diversas plataformas. Ela também possibilita operações de baixo nível, muito úteis em diversos casos. Porém proporciona baixa abstração dos dados e das operações sobre eles, sendo que tarefas como alocação e desalocação de memória devem ser feitas explicitamente. Também oferece uma biblioteca padrão sem os tipos *String* e tipos de coleção. Estas características fazem da linguagem C uma excelente linguagem para trabalhos mais de baixo nível, porém tornam-a difícil e complexa para tarefas que envolvem *Strings* e outros tipos de cadeia de dados, por exemplo, seqüências genéticas.

A Linguagem C++ [Stroustrup 1997] é uma evolução da Linguagem C. Esta linguagem proporciona a utilização da programação Orientada a Objetos como sua principal característica. Outras características, como a compatibilidade, tanto de código fonte como binária com a Linguagem C e uma biblioteca padrão mais completa que sua antecessora, incluindo o tipo *String* e tipos para coleções, tornam-a uma linguagem muito útil e funcional. No entanto, o aprendizado desta linguagem é dificultado pela Orientação a Objetos e pela necessidade de criação e destruição explícita das variáveis utilizadas.

A linguagem Java [Sun 2006] não é uma evolução direta da Linguagem C++, ela foi baseada na Linguagem Smalltalk, porém apresenta diversas similaridades com C++. Entre elas, ressalta-se a sintaxe bastante similar e a Orientação a Objetos, sendo que em Java a Orientação a Objetos é obrigatória, enquanto em C++ é opcional. Entre as características da Linguagem Java, pode-se destacar a compatibilidade binária entre diferentes plataformas, não ser necessário a declaração explícita de liberação de memória e uma biblioteca padrão muito completa. As principais dificuldades da Linguagem Java é a dificuldade do seu aprendizado e é uma linguagem se comparada com as demais aqui apresentadas, necessita muitos recursos computacionais.

Provavelmente a linguagem mais utilizada para a criação de tarefas na bioinformática é a Linguagem Perl [O'Reilly 2006]. Possuir tipos abstratos úteis e operação eficientes já inclusas na linguagem é uma das principais características dela. Entre os tipos já inclusos, pode-se destacar *Strings* com suas expressões regulares, onde é possível executar complexas operações em cadeias de caracteres em apenas uma linha de código. Outra característica importante é a não necessidade de declaração dos tipos das variáveis e destruição delas, sendo tudo feito implicitamente. A utilização do paradigma de programação *procedural*, mais simples que o Orientado a Objetos, junto com as características citadas anteriormente ajudam a fazer da Linguagem Perl uma linguagem de fácil utilização. Porém há uma falta de organização nos códigos fontes desta linguagem,

tornando-a confusa para a criação de projetos de maior porte.

Após a análise das linguagens de programação mais utilizadas, concluiu-se os seguintes aspectos:

- Tipos Abstratos de Dados auxiliam no aprendizado e aumentam a produtividade de uma linguagem;
- Orientação a Objetos é útil, porém complexa para o usuário fora da área da computação;
- Variáveis fortemente tipadas tornam o código maior, porém auxiliam na legibilidade;
- Alocação e desalocação explícita de memória tornam a linguagem difícil e prejudicam a produtividade;
- A execução em diversas plataformas, incluindo as com baixo poder computacional, ajuda a expandir o uso da linguagem de programação.

Utilizando estes cinco aspectos, junto com entrevistas com pesquisadores da bioinformática, foram definidos os requisitos funcionais e não funcionais da linguagem BioTongue, que são discutidos na seção seguinte.

3.2. Requisitos não funcionais e funcionais

O objetivo destes requisitos é ter uma lista de prioridades para a linguagem e planejar o modo como o compilador e a máquina virtual da mesma serão desenvolvidas. Os requisitos são divididos em duas partes, os não funcionais, que são questões de implementação, velocidade e portabilidade e os requisitos funcionais são as funcionalidades que a linguagem deverá possuir.

O principal objetivo dos requisitos não funcionais é a portabilidade do compilador e da máquina virtual. Eles são apresentados na listagem a seguir.

1. Deverão ser implementados na linguagem C, respeitando o padrão ANSI C;
2. Deverão compilar e executar em todas as plataformas que possuam um compilador compatível com o padrão ANSI C;
3. Deverão ocupar pouco espaço na memória, junto com as suas bibliotecas padrão.

Os requisitos funcionais são apresentados na listagem a seguir. Este requisitos possuem como finalidade apresentar as funcionalidades da linguagem para o usuário. Para melhor organização do projeto e implementação, eles estão divididos em grupos: os quatro primeiros requisitos estão relacionados aos Tipos Abstratos de Dados; o quinto e o sexto requisito referem a definição e utilização de variáveis; o sétimo requisitos é sobre a estrutura do código fonte da BioTongue.

1. Utilizar-se Tipos Abstratos de Dados que devem abstrair totalmente os detalhes da plataforma;
2. Possuir embutida tipos abstratos para *Strings*, Genética Molecular e Coleções;
3. Possuir nos Tipos Abstratos de Dados operações que auxiliam na sua utilização;
4. Existir uma biblioteca padrão contendo funcionalidades para entrada e saída e trabalho com *Strings*;
5. A máquina virtual possuirá um *garbage collector* para tornar desnecessário a desalocação explícita de memória;
6. Definir os tipos das variáveis no momento de criação;
7. Diferenciação nos arquivos fontes que serão executados e das bibliotecas da linguagem.

4. A Linguagem BioTongue

Nesta seção é apresentada a Linguagem BioTongue. Primeiramente é apresentada a estrutura da linguagem, depois os tipos abstratos de dados e suas operações e por fim as instruções de iteração e condicionalidade.

4.1. Estrutura da Linguagem BioTongue

A linguagem BioTongue utiliza a palavra reservada *job* para definir que o arquivo fonte é uma tarefa, ou seja, deve ser executado pela máquina virtual. Também é utilizada a palavra reservada *lib* para arquivos que serão bibliotecas, ou seja, os arquivos onde estarão definidos os Tipos Abstratos de Dados e suas operações. A Figura 1 apresenta a *BNF* da Linguagem BioTongue.

Na Figura 2 é exibido um típico "Hello World" na Linguagem BioTongue. A primeira linha define que este código fonte é um *job* chamado *example*. Na linha 3, uma literal é enviada para saída padrão e na linha 5 o *job* é finalizado.

A opção por utilizar arquivos separados para os códigos fontes executáveis e de bibliotecas tem como objetivo favorecer a organização do código fonte de projetos maiores e proporcionar a reusabilidade de código. Na figura 3 é exibido um exemplo de *lib*. Nesta *lib* um novo tipo chamado *SequenceInfo* é criado. Mais informações sobre a criação de tipos é apresentado na seção 4.2.

4.2. Tipos Abstratos de Dados

Como citado na seção 3.2, a linguagem BioTongue utiliza o conceito de Tipos Abstratos de Dados como modelo para a abstração das informações. A principal idéia dos Tipos Abstratos de Dados é que o tipo é caracterizado pelas operações que se realizam nele, ou seja, é caracterizado pela sua aplicação na linguagem e não pela sua implementação, sendo ela oculta para o usuário. Linguagens como C e C++ implementam tipos através de *struct*, porém esta técnica não oculta a implementação. A Orientação a Objetos permite ocultar os detalhes de implementação do usuário e proporciona outras operações sobre os objetos, como por exemplo, a herança. O conceito de Tipos Abstratos de Dados é um meio termo entre as estruturas e a orientação a objetos.

Na Linguagem BioTongue os tipos podem ser divididos em Básicos, Genéticos e Coleções. Os tipos Básicos são o *Numeral*, representando valores numéricos, o *Boolean* que representa verdadeiro ou falso e o *String*, representando cadeias de caracteres.

Os tipos Genéticos, fornecem abstrações para os tipos da biologia molecular. Eles são os seguintes tipos: *Base*, *Codon*, *Aminoacid*, *Sequence* e *Protein*. O tipo *Base* representa as bases nucleicas que compõem as seqüências de *DNA* e *RNA*. As bases podem ser: *A* (Adenina), *C* (Timina), *G* (Guanina), *T* (Timina) e *U* (Uracila), sendo que as seqüências de *DNA* são formadas pelas quatro primeiras e nas de *RNA* a Timina é substituída pela Uracila. O tipo *Codon* é uma trinca de bases que pode ser transformada em aminoácidos. Os aminoácidos são representados pelo tipo *Aminoacid* e são ao todo vinte. A tabela completa com os nomes dos aminoácidos e as siglas utilizadas no BioTongue estão presentes no manual da linguagem¹. Por fim, as seqüências de *DNA* ou *RNA* são representadas pelo

¹Disponível em <http://albrecht.rtfm.com.br/biotongue/>

```

<source>: <source_type> IDENTIFIER begin
         <source_body>
         end <source_type> .

<source_type>: job | lib

<source_body>: <source_body> <body> ; | <body> ;

<body>: <declaration> | <statement>

<declaration>: <type_decl> | <var_decl> | <op_decl>

<statement>: <expression_stmt> | <iteration_stmt>
            | <conditional_stmt> | <return_stmt>

<type_decl>: IDENTIFIER is ADT begin
            <attr_list>
            end ADT

<attr_list>: <attr_list> ; <var_decl> | <var_decl>

<op_decl>: <type> IDENTIFIER operation in <type> ( <param_list> ) begin
         <body>
         end operation

<param_list>: <param_list> , <var_decl> | <var_decl>

<expression_stmt>: <expression_stmt> <relational> <expression_stmt>
                 | <complex_expr> | <type> | default | Nil

<relational>: <- | -> | > | => | < | <= | == | <>

<complex_expr>: <complex_expr> <add> <term> | <term>

<add>: + | -

<term>: <term> <multiply> <factor> | <factor>

<multiply>: * | /

<factor>: ( <expression_stmt> )
        | IDENTIFIER
        | <operation_call>
        | LITERAL
        | CONSTANT

<operation_call>: <complex_id>( <args> )

<complex_id>: <complex_id> . IDENTIFIER | IDENTIFIER

<args>: <args> , <expression_stmt> | <expression_stmt>

<iteration_stmt>: <foreach_stmt>
               | <while_stmt>

<foreach_stmt>: foreach ( <var_decl> in IDENTIFIER ) begin <body> end foreach

<while_stmt>: while ( <expression_stmt> ) begin <body> end while

<conditional_stmt>: if ( <expression_stmt> ) begin <body> end if
                  | if ( <expression_stmt> ) begin <body> else <body> end if

<var_decl>: IDENTIFIER in <type>

<type> : IDENTIFIER
       | Boolean | String | Numeral
       | Base | Codon | Aminoacid | Sequence | Protein
       | List | HashTable | Matrix

```

Figure 1. *Bnf* da Linguagem BioTongue.

```

1  job example begin
2
3     "Olá Mundo!" -> default;
4
5  end job.

```

Figure 2. Exemplo de um *job* em BioTongue.

```

1  lib example begin
2
3     SequenceInfo is ADT begin
4         sequence is Sequence;
5         value is Numeral;
6     end ADT;
7
8  end lib.

```

Figure 3. Exemplo de uma *lib* em BioTongue.

tipo *Sequence* e as proteínas, formadas por uma seqüência de aminoácidos são representadas pelo tipo *Protein*.

São três os tipos Coleção: o tipo lista (*List*), o tipo Tabela *Hash* (*HashTable*) e o tipo Matriz (*Matrix*). O tipo *List* oferece uma lista duplamente encadeada, com operações para adição, remoção, busca de elementos, concatenação e intercessão. O tipo *HashTable* é uma tabela *Hash* que, segundo [Cormen et al. 1989], oferece a vantagem de armazenar e recuperar informação em tempo constante $O(1)$. O tipo *Matrix* pode ser utilizado como um *Array* n-dimensional de elementos ou também como uma matriz para operações numéricas. Quando o tipo agregado da *Matrix* for *Numeral*, a linguagem fornece operações como multiplicação, soma e inversa.

Além dos tipos apresentados anteriormente, é possível criar novos tipos utilizando a palavra reservada *ADT*, sigla para *Abstract Data Type* (Tipo Abstrato de Dado). Exemplo de criação de um novo tipo, chamado *Info* e que contém os atributos *name* e *sequence*, é exibido no trecho de código da Figura 4.

```

1  Info is ADT begin
2     name      is String;
3     sequence is Sequence;
4  end ADT;

```

Figure 4. Criação de um novo tipo abstrato.

4.2.1. Utilização dos Tipos Abstratos de Dados

Para utilizar os Tipos Abstratos de Dados, devem ser criadas variáveis. Quando uma nova variável é criada, o seu conteúdo é automaticamente iniciado como *Nil* (nulo). É permitido efetuar operações sobre variáveis vazias, porém até que a variável receba algum conteúdo, a exibição dela será *Nil*.

A atribuição de valores a uma variável pode ser feita com dois operadores: o operador *de-para* e o operador *para-de*. No código fonte exibido na figura 5 é apresentado um exemplo de criação e atribuição de variáveis.

```
1 sequence is Sequence;
2 value is Numeral;
3
4 value <- default;
5 sequence <- "ACCTGG";
6 sequence -> default;
7 value -> default;
```

Figure 5. Criação e atribuição de variável.

A palavra reservada *is* serve para informar o tipo de cada identificador. No exemplo acima, a primeira linha informa que *sequence* é um tipo *Sequence* e na segunda linha que *value* é um tipo *Numeral*. Na quarta e quinta linha, a atribuição *para-de* é utilizada para iniciar o valor da variável *value* com o valor lido da entrada padrão e a variável *sequence* com uma literal. Em ambos os casos, o valor lido é convertido implicitamente para o tipo da variável. Nas últimas duas linhas, a atribuição *para-de* é utilizada para que a representação das variáveis *sequence* e *value* sejam atribuídas para a saída padrão.

A palavra reservada *default* é um *alias* para a entrada e saída padrão. Na versão atual do BioTongue, elas são a entrada e saída do console de texto. Em versões futuras pretende-se definir o que ela será, por exemplo, um arquivo ou um *socket* para troca de informações na *Internet*. Para leitura da entrada padrão, deve-se atribuir uma variável para ler o conteúdo digitado. Se a variável que recebeu o valor for um tipo Básico ou Genético, a conversão é feita implicitamente, caso contrário o usuário deverá tratar o valor lido. Para escrita na saída padrão, conforme mostrado na Figura 5, deve-se atribuir a variável à saída padrão e a sua representação em *String* será exibida nela.

4.2.2. Hierarquia dos tipos genéticos

Os tipos genéticos da Linguagem BioTongue possuem uma hierarquia que segue a Genética Molecular. Esta hierarquia define quais tipos estão agregados em outros e também as transformações que podem ser efetuadas.

Como BioTongue é uma linguagem tipada, sendo que cada variável só pode conter tipo que foi declarada e *casts* não são permitidos, decidiu-se adotar uma hierarquia para facilitar a operação com os tipos genéticos. O objetivo desta hierarquia é permitir que as operações efetuadas em variáveis genéticas funcionem de forma transparente na linguagem. Uma operação permitida é atribuir o conteúdo de um *Codon* a um *Aminoacid*, de tal forma que nesta atribuição o valor é convertido de forma transparente. Pode-se obter os *Codon* de um *Sequence* e deles, obter os seus *Base*. A Tabela 1 apresenta quais tipos de conversões e agregações são válidas.

Resumindo, a utilidade de haver uma hierarquia de tipos genético é a abstração do conteúdo das variáveis, das conversões e a obtenção de variáveis que estão agregadas em outras, desta forma, resultando em maior facilidade de aprendizado da linguagem e maior

Table 1. Hierarquia dos tipos genéticos da Linguagem BioTongue

Nome do Tipo	Nome Genético	Agrega	Converte-se
<i>Base</i>	Base Nucléica	-	-
<i>Codon</i>	Códom	<i>Base</i>	<i>Aminoacid</i>
<i>Sequence</i>	Seqüência	<i>Base, Codon</i>	<i>Protein</i>
<i>Aminoacid</i>	Aminoácido	-	-
<i>Protein</i>	Proteína	<i>Aminoacid</i>	-

produtividade e regibilidade do código fonte. Um exemplo de conversões implícitas e utilização de operadores é apresentado na Figura 6.

```
1  job converter begin
2
3      sequence          is Sequence;
4      aminoAcid        is Aminoacid;
5      codon_1, codon_2 is Codon;
6
7      codon_1    <- "ACT";
8      codon_2    <- "CTG";
9      sequence   <- codon_1 + codon_2 + "A";
10     aminoAcid  <- codon_1;
11
12     sequence   -> default;
13     _newline_  -> default;
14     aminoAcid  -> default;
15
16 end job.
```

Figure 6. Conversão de tipos no BioTongue.

No exemplo apresentado na Figura 6, as 5 primeiras linhas são definição do *job* e as declarações de variáveis. As duas variáveis do tipo *Codon* são iniciadas nas linhas 7 e 8. Na linha 9 há uma concatenação entre dois códoms e uma literal e o resultado desta concatenação é atribuído à variável *sequence*. Neste caso, a literal é tratada como um *Base*. Mas dependendo do contexto, ela poderia ser tratada como um *Aminoacid*. Caso a literal tivesse três caracteres de tamanho seria tratada como um *Codon* e caso fosse maior, será tratada como *Sequence*. Dependendo do contexto, literais com mais de um caractere poderão ser interpretadas com *Protein*. A variável *sequence* é exibida na saída padrão na linha 12. Na linha 13, uma variável interna (*_newline_*) da linguagem que representa nova linha é exibida na saída padrão. Na linha 14 ocorre uma conversão implícita de *Codon* para *Aminoacid*. É exibido na Figura 7 a saída da execução do fonte da Figura 6.

```
ACTCTGA
Thr
```

Figure 7. Saída da execução do exemplo da Figura 6.

4.2.3. Operações dos Tipos Abstratos de Dados

Como dito nas seções anteriores, a utilização dos Tipos Abstratos é feita através das operações. Estas operações podem ser feitas explicitamente, por meio de *tipo.operacao()* ou por meio de operadores, por exemplo o operador de adição (+). Todos os Tipos da Linguagem BioTongue possuem ao menos quatro operações: *variavel.type()* que retorna o tipo da variável, *variavel.isNil()* que retorna se o conteúdo da variável está vazio, *variavel.attributes()* que retorna um *List* de todos os atributos do tipo da variável e a *variavel.operations()* que retorna um *List* de todas as operações do tipo da variável.

Na linha 9 do exemplo da Figura 6, observa-se uma concatenação entre três variáveis. A primeira concatenação, entre os dois códons, utiliza implicitamente a operação *concatenate* do tipo *Codon*. Neste tipo, esta operação sempre retornará um *Sequence*. A concatenação seguinte, entre o *Sequence* retornado da concatenação anterior e um *String* é feito utilizando a operação *concatenate* do tipo *Sequence* e retorna como resultado uma *Sequence*. Caso a *String* contesse algum caractere inválido para seqüências genéticas, um erro de execução seria gerado.

São diversas as operações dos tipos internos da linguagem, todas criadas com o objetivo de tornar a linguagem eficiente e simples de utilizar. A melhor maneira de conhecer todas as características da linguagem é ler o manual da linguagem.

4.2.4. Criação de novas operações

Além das operações existentes nos Tipos Abstratos de Dados, é possível expandir os tipos adicionando-os novas operações. Na Figura 8 é exibido um trecho de código onde uma nova operação é criada para o tipo *Info*, que é definido no exemplo da Figura 4.

```
1 Info create is operation in Info(n is String,  
                                s is Sequence) begin  
2   i is Info;  
3   i.sequence <- s;  
4   i.name <- n;  
5  
6   return i;  
7 end operation;
```

Figure 8. Criação de operações.

A primeira palavra da declaração de operação é o tipo do retorno dela. Seguidamente é dito que *create* é uma operação no tipo *Info*. Da linha 2 até a 4, a variável é criada e seus conteúdos iniciado. Na linha 6 a variável é retornada e a operação é finalizada. Na linha 7 a declaração da operação é terminada. Nas operações pode-se executar todos os comandos, com exceção de definição de novos tipos e funções.

4.3. Instruções de iteração e condicionalidade

A Linguagem BioTongue, assim como as linguagens de programação apresentadas anteriormente, possui instruções de iteração e condicional. A instrução condicional é o *if* e as

de iteração são o *while* e o *foreach*. Na Figura 9 é exibido um trecho de código que utiliza estas três instruções.

```
1 s is Sequence;
2
3 sequence <- default;
4
5 while (sequence <> Nil) begin
6     foreach(codom is Codon in sequence) begin
7         if (codom.toProtein() == "Thr") begin
8             "Treonina encontrada" -> default;
9         end if;
10    end foreach;
11    s <- default;
12 end while;
```

Figure 9. Uso de instruções de iteração e condicionais

No exemplo, enquanto a variável *sequence* ler algo, ou seja, diferente de *Nil*, a execução ficará amarrada dentro do *while*. Na linha 6, a instrução *foreach* itera sobre todos os *Codon* da variável *sequence*. Para construção dessa iteração no conteúdo da variável, foi seguido a hierarquia apresentada na Tabela 1, onde diz que o tipo *Sequence* agrega *Codon* e *Base*. Na linha 7 o *codon* é convertido para um *Aminoacid* e comparado ao aminoácido *Treonina*. Caso a expressão seja verdadeira, uma mensagem é exibida na saída padrão. As linhas seguintes são o fechamento das instruções e a leitura da entrada padrão para a variável *sequence*.

5. Desenvolvimento do Compilador e da Máquina Virtual

O modelo utilizado no desenvolvimento da Linguagem BioTongue foi o Iterativo Incremental. Este modelo foi aplicado da seguinte forma: Metas são definidas, normalmente são novas funcionalidades ou melhorias na máquina virtual. Através delas são criados códigos que devem ser executados corretamente. Então implementa-se estas funcionalidades no compilador e na máquina virtual e o resultado é testado executando os códigos fontes criados para a meta. No fim da implementação da nova funcionalidade, estes códigos fontes são guardados para execução de testes futuros.

O compilador e a máquina virtual da Linguagem BioTongue são desenvolvida utilizando a Linguagem C. Os motivos para esta escolha estão presentes na seção 3.1. O Analisador Léxico e Sintático são desenvolvidos utilizando as ferramentas *Flex*² e *Bison*³ respectivamente. Ambas ferramentas são compatíveis com as ferramentas *Lex* e *Yacc*. A opção por não desenvolver os próprios analisadores léxicos e sintáticos ocorreu porque a sintaxe da linguagem não esta completamente definida e diversos aperfeiçoamentos continuam sendo feitos.

5.1. Máquina Virtual

A Linguagem BioTongue é uma linguagem interpretada e as instruções para a máquina virtual são criadas em tempo de execução, ou seja, não é gerado código. O analisador

²Disponível em <http://www.gnu.org/software/flex/>

³Disponível em <http://www.gnu.org/software/bison/>

sintático cria estruturas com as informações a serem executadas e passa-as para a máquina virtual onde serão executadas. Esta opção foi feita para facilitar o desenvolvimento inicial. Em futuras versões do BioTongue pretende criar um compilador que gere código para ser executado na máquina virtual.

5.2. Estruturas internas

A implementação da máquina virtual da Linguagem BioTongue possui a estrutura *biotongue_t* como estrutura principal da execução. Este tipo possui os atributos necessários para a execução: memória de dados e execução, a tarefa sendo executada, bibliotecas e outras informações.

Os tipos da Linguagem BioTongue são internamente uma estrutura *bt_type_t*. Os atributos principais desta estrutura são o *BtType type* que informa qual é o tipo, *bt_hash_t attrs* e *bt_hash_topers* que são respectivamente tabelas *hash* contendo os atributos e as operações do tipo. O atributo *void* content* é um ponteiro para a estrutura que contém as informações específicas do tipo. Por exemplo, se for um tipo *String* apontará para a uma variável do tipo *bt_string_t*, se for *Sequence*, apontará para uma variável *bt_sequence_t* e se for um tipo definido pelo usuário, apontará para um *bt_newtype_t*.

As instruções de iteração e condicional são implementadas como funções na máquina virtual. Por exemplo a instrução *foreach* é executado pela função *exec_foreach_stmt* cujo recebe como parâmetro uma estrutura *stmt_foreach_t*. Esta estrutura contém a variável que receberá o valor, a variável que possui o conteúdo a ser iterado e a lista dos comandos que estão amarrados a instrução *foreach*.

A memória da máquina virtual é implementada numa tabela *hash*. A estrutura desta tabela *hash* é a mesma dos atributos *attrs* e *opers* da estrutura *bt_type_t* e também da estrutura interna do tipo *HashTable*. Na memória, variáveis, tipos definidos pelo usuário e de bibliotecas são armazenados. A máquina virtual do BioTongue possui outra memória que é a pilha de execução. Nesta memória são armazenados os parâmetros para as operações e seus retornos.

6. Considerações finais

Neste artigo foi apresentada a Linguagem de Programação BioTongue. Ela possui como importante característica a abstração de tipos genéticos e a possibilidade de adicionar novas operações a tipos já existentes. Destaca-se também as conversões transparentes entre tipos da linguagem e uma sintaxe simples e ortogonal. Resumindo, é uma linguagem planejada e implementada para oferecer o máximo de funcionalidade e facilidade para trabalhos na área de bioinformática, tanto para usuários da área da ciência da computação quanto para da biologia e genética molecular.

No momento a Linguagem BioTongue está funcional, porém sem todas as funcionalidades apresentadas neste artigo. O desenvolvimento dela pode ser acompanhado em <http://colla.albrecht.rtfm.com.br>, seu código fonte pode ser obtido via SubVersion⁴ em <http://svn.albrecht.rtfm.com.br> e o manual e as informações do estado atual do compilador e da máquina virtual em <http://albrecht.rtfm.com.br/biotongue/>. Colaborações de todos os tipos são bem vindas.

⁴Disponível em <http://subversion.tigris.org/>

References

- Cormen, H. T. ; Leiserson, C. E. e Rivest, R. L. (1989). Introduction to Algorithms. *The MIT electrical engineering and computer science series*.
- Durbin, R. ; Eddy, S. ; Krogh, A. e Mitchison, G. (1998). Biological sequence analysis: Probabilistic models of proteins and nucleic acid. *Cambridge University Press*, 8th edition.
- Kernighan, B. W. e Ritchie, D. (1997). The C Programming Language. *Prentice Hall PTR*, 2nd edition.
- Lewin, B. (2001). Genes VII *Artmed Editora*.
- Mount, D. W. (2004). Bioinformatics: Sequence and Genome Analysis. *Cold Spring Harbor Laboratory Press*, 2th edition.
- O'Reilly Media, Inc. (2006). Perl.com: The source for Perl. <http://www.perl.com/> .
- Strachan, T. ; Read, A. P. (2002). Genetica Molecular Humana. *Artmed Editora*.
- Stroustrup, B. (1997). The C++ Programming Language. *Addison-Wesley Professional*, 3rd edition.
- Sun Microsystems, Inc. (2006). Java Technology. <http://java.sun.com/> .