Processamento de Linguagem Natural Controlada utilizando uma Máquina de Moore

Controlled Natural Language Processing using a Moore Machine

Hortevan M. Frutuoso[1], Benjamin R. C. Bedregal[2]

[1]PPGSC – Programa de Pós-Graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte (UFRN)– 59.078-970 – Natal – RN – Brasil

[2]DIMAP – Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte (UFRN) – 59.078-970 – Natal – RN – Brasil

hortevan.mf@gmail.com, bedregal@dimap.ufrn.br

Abstract. *This paper proposes an algorithm for controlled natural language processing using a Moore Machine. Initially, a brief description of the concepts involved, such as chatterbot, automata, and ontology, will be presented. Then, an algorithm for controlled and expandable natural language processing will be presented. The algorithm works as a translator that transforms the sentence in natural language informed into executable code to a stack-based virtual machine, executing the processing according to the knowledge stored in the automaton.*

Resumo. *Este artigo propõe um algoritmo para processamento de linguagem natural controlada usando uma Máquina de Moore. Inicialmente será apresentada uma breve descrição dos conceitos envolvidos, tais como chatterbot, autômatos e ontologia. Em seguida será apresentado um algoritmo para processamento de linguagem natural controlada e expansível. O algoritmo funciona como um tradutor que transforma a sentença em linguagem natural informada em código executável para uma máquina virtual baseada em pilha, executando o processamento tal qual o conhecimento armazenado no autômato.*

1. Introdução

Computadores são máquinas de processamento de informação. Eles funcionam segundo instruções contidas em programas armazenados em sua memória. Esses programas são codificados em uma linguagem numérica composta por combinações de 0's e 1's e adequadamente chamada de linguagem de máquina. Os seres humanos, por sua vez, se comunicam através de uma linguagem espontânea chamada "Linguagem natural". Linguagens de programação como Java, por exemplo, são utilizadas para, grosso modo, passar conhecimento ao computador. Essas linguagens possuem comandos e estruturas que se assemelham à linguagem natural e dizem ao computador como algo deve ser feito. Entretanto, mesmo as linguagens de altíssimo nível estão consideravelmente longe da linguagem usada pelos seres humanos. Pesquisas têm sido feitas com o intuito de

diminuir essa distância e tornar a comunicação homem-máquina mais próxima daquilo que chamamos de natural. Nesse contexto surge então o Processamento de Linguagens Naturais (PLN) como uma subárea da Inteligência Artificial cujo objetivo é fazer o computador reagir adequadamente à linguagem humana. Atualmente, existem diversos softwares utilizados para processar uma linguagem natural. Tais como: os Assistentes Virtuais e os *Chatterbots*.

Neste artigo serão abordados temas como softwares de conversação e linguagem natural controlada. Em seguida será apresentado um algoritmo proposto para processar sentenças de uma linguagem natural controlada, de forma expansível, utilizando-se de uma base de conhecimento representada por uma Máquina de Moore. O objetivo deste algoritmo é, a partir do conhecimento armazenado, reagir ao usuário executando as ações adequadas com intuito de atender de forma adequada às sentenças passadas pelo usuário.

2. Conceitos Teóricos

Linguagem natural (LN) é aquela que surge espontaneamente no meio social [Lancaster 2003]. É através dela que os seres humanos trocam informações. Seja de forma oral, escrita ou até mesmo pictográfica, a correta compreensão de uma mensagem depende de vários fatores. Uma única expressão em linguagem natural pode ter significados tão diferentes, quanto variam o tempo, o espaço e o conhecimento de mundo dos participantes de um diálogo. Ela é altamente ambígua. E embora grandes avanços tenham acontecidos no campo da Inteligência Artificial (IA), computadores ainda têm dificuldades em processá-la. De fato, a complexidade inerente ao trato computacional da linguagem natural deve-se principalmente a sua grande ambiguidade. Longe de ser uma tarefa trivial, o PLN deve permitir ao computador "compreender" a linguagem natural e também apresentar uma resposta coerente para o usuário.

Existem diversas abordagens para o PLN. Contudo, neste artigo será tratada apenas a Análise de texto. [Schneider 2001] afirma que analisar textos é obter conhecimento ou informação a partir de textos. Segundo [Neto, Tonin e Prietch 2010], existem basicamente duas correntes de aplicações na análise textual: Aplicações baseadas em texto e aplicações baseadas em diálogos. A primeira consiste em obter informações a partir dos dados contidos em textos ou base de dados textuais. Já a segunda se utiliza das sentenças obtidas diretamente do usuário através da simulação de um diálogo. Assim, a partir das indagações feitas pelo usuário, o computador deve descobrir qual a resposta mais provável; de forma a simular uma conversa. Não se trata de meramente encontrar documentos ou partes do texto dentro de grandes bases de dados; mas extrair informações a partir das cadeias de léxicos que compõem o texto e processá-las de forma que o software ofereça ao usuário uma resposta que faça sentido. Os sistemas mais famosos desta categoria são os *chatterbots*.

2.1 Softwares de conversação: *chatterbots*

[Comarella e Café 2008] definem *chatterbot* como um software capaz de se passar por um ser humano em um diálogo textual. *Chatterbots* (literalmente, "robôs de conversa") são softwares capazes de responder a perguntas feitas em linguagem natural, dentro de um contexto específico. Eliza foi o primeiro software do gênero. A tecnologia utilizada

no Eliza era bem simples, baseada apenas no casamento dos padrões extraídos das palavras empregadas no texto do usuário e a base de conhecimento do sistema. Mais informações sobre Eliza e outros *chatterbots*, bem como o estado da arte, podem ser encontradas em [Alencar 2011], [Comarella e Café 2008] e [Neves e Barros 2005]. A título de curiosidade, no Brasil, destaca-se o trabalho desenvolvido por [Primo e Coelho 2002], que foi o primeiro *chatterbot* em português.

2.2 Linguagem Natural Controlada e autômatos

A Tese de *Church-Turing* afirma que toda função que é naturalmente considerada computável pode ser computada através de uma Máquina de Turing. Embora não se possa provar formalmente, até o presente momento não se conhece outro modelo computacional mais completo que a Máquina de Turing. Ela é capaz de processar linguagens pertencentes ao conjunto mais abrangente da Hierarquia de *Chomsky*: O conjunto "Tipo 0" ou conjunto das Linguagens Enumeráveis Recursivamente [Menezes 2008]. Entretanto, uma LN não é uma linguagem formal. Ela surge de forma espontânea e não através de definições matemáticas precisas, o que a coloca fora do modelo de *Chomsky*. [Jager e Rogers 2012] afirmam que durante décadas pesquisadores debateram sobre a possível localização da LN dentro do modelo de *Chomsky*. Porém, o resultado final é inconclusivo. O fato é que, até o presente momento, o mais poderoso modelo computacional não foi capaz de reconhecer completamente a LN. Uma discursão mais detalhada sobre a LN e a Hierarquia de *Chomsky* pode ser vista em [Kornai 1985], [Jager e Rogers 2012]. Apesar de não ser possível reconhecer completamente a LN, ao separar-se um subconjunto dela, de forma controlada, pode-se obter uma linguagem tão simples quanto uma linguagem livre de contexto ou até mesmo uma linguagem regular – que são as classes de linguagem menos abrangentes da Hierarquia de *Chomsky*. A esse subconjunto dá-se o nome de Linguagem Natural Controlada (LNC).

[Kuhn e Schwitter 2008] definem Linguagem Natural Controlada (LNC) como um subconjunto da linguagem natural que foi reduzido a fim de eliminar ou pelo menos diminuir a ambiguidade da linguagem. Do ponto de vista prático, consiste em um conjunto de palavras (ou símbolos) e algumas regras de produção que definem como esses símbolos podem ser utilizados para formar sentenças válidas. O Português Controlado, por exemplo, é um subconjunto do português natural. Com uma LNC é possível definir regras de transição simples que permitam formar frases específicas, conforme a necessidade. Softwares como o Cortana, por exemplo, fazem uso de uma LNC para receber ordens e executar certos comandos, tais como: "que horas são?" ou "qual a previsão do tempo?". Construções deste tipo podem ser tratadas como pertencentes ao universo das linguagens regulares – facilmente reconhecida por Autômatos Finitos. Autômatos Finitos e autômatos com pilha são modelos computacionais abstratos da Teoria da Computação utilizados no reconhecimento de linguagens regulares e livres de contexto, respectivamente. Um autômato é alimentado por uma fita contendo uma sequência de símbolos pertencentes a um alfabeto. A cada estado, um símbolo é lido na fita e então uma regra de transição é acionada conforme as regras definidas para aquele estado. Assim, o próximo estado é determinado pelo estado atual e pelo símbolo lido na fita. Ao término da sentença contida na fita, verifica-se se o último estado é um estado válido para saída – ou seja, um estado final ou de aceitação. Sequências que – a partir do estado inicial – encerrem-se em um estado final; são ditas

reconhecidas ou aceitas pelo autômato. Caso a fita termine em um estado não-final, a sentença foi rejeitada e portanto não pertence à linguagem reconhecida pelo autômato. Diz-se que um autômato é determinístico quando cada símbolo de entrada leva a um único estado possível – ao contrário do não-determinístico. Mais informações sobre a teoria dos autômatos podem ser encontradas em [Aho, Sethi e Ullman 1995], [Bedregal, Acióly e Lyra 2010].

Os Autômatos com Pilha (AP), como o nome sugere, possuem uma pilha para auxiliar na análise da fita e são utilizados para reconhecimento de linguagens livres de contexto. Assim como autômatos finitos podem ser determinísticos ou não-determinísticos, AP's também podem ser determinísticos (APD) ou não-determinísticos (APN). No entanto, diferentemente do caso de autômatos finitos, os APD's são menos poderosos que os APN's, no sentido que a classe das linguagens reconhecidas por APD's (chamadas de livres de contexto determinísticas) são uma subclasse das linguagens livres de contexto, que é a classe das linguagens reconhecidas por APN's. APN's podem ser usados na formalização sintática das linguagens de programação de alto nível (em seus diversos paradigmas: orientado a objetos, funcional, procedural etc.).

Uma LNC pode ter estruturas mais complexas, como em uma linguagem de programação. No entanto, inserir nela construções linguísticas avançadas pode levar a um gigantesco aumento da complexidade e da ambiguidade. O que contraria o objetivo principal da LNC – que é ter uma estrutura linguística livre – tal como se fala naturalmente. LNC's são criadas com o objetivo de permitir o uso de sentenças de forma livre e até mesmo redundantes. Como ela é um subconjunto da linguagem natural, a LNC permite que duas ou mais sentenças diferentes tenham um mesmo significado semântico – mesmo que utilizem símbolos diferentes em sua construção. Por outro lado, o uso de estruturas mais avançadas permite construções linguísticas poderosas, tais como laços de repetição e estruturas de controle – que são bastante úteis ao se passar instruções a um computador. Se o objetivo é fazer uso dessas estruturas, uma LNC pode não ser adequada. Acrescente-se ainda o fato de que ao inserir tais estruturas em uma LNC corre-se o risco de fazer com que ela deixe de ser uma linguagem natural.

2.3 Máquinas de Moore

Reconhecer a linguagem é apenas parte da solução. Para que o computador possa processar corretamente a cadeia em análise, o computador deve ser capaz de reagir de forma a representar cada diferente estado possível. Ou seja, deve executar algum código que faça uma alteração no comportamento da máquina, de forma a refletir cada estado atingido no autômato. Assim, ao invés de se utilizar autômato finito, pode-se usar Máquinas de Moore para o PLN.

[Menezes 2008] define uma Máquina de Moore (MM) como um Autômato Finito Determinístico (AFD) onde cada estado possui uma palavra de saída associada. Partindo-se do estado inicial, a cada novo estado atingido uma nova cadeia de palavras é acrescentada à cadeia de saída, de forma que cada sentença de entrada gera uma nova sentença de saída. Se essa sentença de saída for um código executável, o computador pode mudar o seu comportamento de forma a refletir os estados alcançados durante a análise da sentença.

2.4 Análise Léxica

A análise léxica consiste em, partindo de uma sentença (cadeia de caracteres) de entrada, produzir uma sequência de símbolos léxicos (*tokens*) para análise posterior. É a base para o funcionamento dos compiladores. Os símbolos léxicos gerados referenciam os caracteres contidos na sentença e lhes dão um significado útil. Como por exemplo: número, constante, palavra reservada, entre outros [Aho, Sethi e Ullman 1995].

2.5 Ontologias

[Gruber 1993] define ontologia como uma estrutura que relaciona conceitos representados através de classes, funções, e outros objetos. É uma forma de representação do conhecimento em forma de grafo que permite armazenar objetos e os relacionamentos entre eles.

3. Processamento de LNC expandida utilizando Máquinas de Moore

Uma MM pode ser usada para detectar a intenção de um usuário expressa em uma LNC. Cabe salientar que a ideia aqui exposta não é criar um software de conversação como um *chatterbot*, mas sim um programa capaz de reconhecer uma sentença passada pelo usuário e responde-la através de uma frase ou ação executada pelo computador. A Figura 1 mostra o diagrama em blocos do algoritmo proposto. A metodologia utilizada para realizar tal processamento será descrita a seguir:

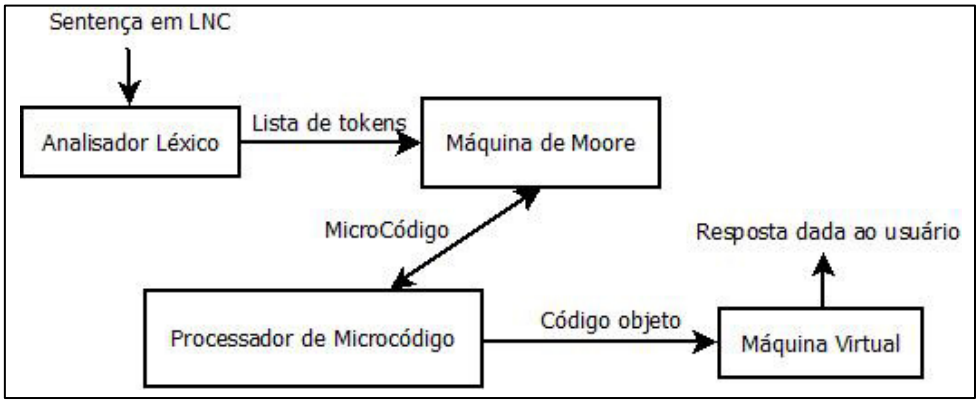

Figura 1. Diagrama em blocos do algoritmo (fonte: do Autor)

Primeiramente, a sentença passada pelo usuário é entregue a um analisador léxico cuja função é identificar na sentença: constantes (texto, número, data, entre outros), operadores e palavras. Foram reservados os valores de 1 a 50 para tipos primitivos, constantes e operadores. Para este trabalho foi definido que o sistema deveria reconhecer cinco tipos primitivos de dados: número, texto, tempo (data e hora), lógico e binário (sequência de bits qualquer). Os códigos de 51 a 100 foram reservados para identificar sentenças mais complexas da linguagem, tais como: comandos, blocos de comandos, tipos etc. Códigos acima de 100 são identificadores de palavras. Cada palavra é indexada através de um dicionário. Ao final da análise, o analisador léxico entregará um vetor de objetos do tipo *Token*. Cada objeto possui um atributo contendo o código do token e outro contendo os caracteres associados ao código *token* encontrado.

A saída do analisador léxico é colocada na fita de uma MM – que nada mais é que uma estrutura tipo lista. A função de transição é uma função parcial – ou seja, ela

não está definida para todos os valores do domínio. Ao receber parâmetros que possuam uma imagem definida, uma transição é realizada. Ou seja, uma transição é feita quando existe, no estado atual, uma aresta acionada pelo elemento lido na fita. Cada estado da MM possui uma lista de arestas contendo o código do token e o estado destino. Quando uma transição é feita, o elemento da fita lido é consumido, deixando o próximo elemento da fita pronto para ser lido. Além dessa lista de arestas, cada estado possui um conjunto de pequenas instruções chamadas microcódigo. Este é dividido em duas partes: a primeira executada ao entrar no estado e segunda ao sair dele. Microcódigos são instruções simples que realizam operações como: saltos, comparações e escrita de código/dados em um *buffer* de saída. O objetivo final do microcódigo é alimentar o *buffer* com o código executável que irá representar a sentença. Para execução desses microcódigos são utilizadas três pilhas: P_1, P_2 para o uso geral e P_3 para recursão no grafo. Enquanto a fita não estiver vazia, sucessivas transições são feitas. A cada transição, zero ou mais instruções do microcódigo são executadas. Quando a fita terminar, caso o último estado presente seja um final, o conteúdo deste *buffer* será enviado para execução na máquina virtual. Para o presente trabalho, foi definida uma máquina virtual baseada em pilha, capaz de reconhecer alguns comandos de comparação, saltos e operações lógico-aritméticas sobre a pilha, entre outros. Possui também instruções avançadas de interação com o usuário, manipulação de dados e comandos que alteram a própria estrutura da MM – excluindo ou criando novos caminhos.

A Figura 2 mostra a parte da MM responsável por tratar expressões lógico-aritméticas. O estado V_0 é o estado inicial. V_1 e V_8 são estados finais. Juntos, os microcódigos contidos nos vértices V_1, V_2, V_3 e V_4 executam o processamento para conversão do conteúdo da fita – que está na notação infixa – para a notação pós-fixa. O microcódigo contido na saída do estado V_1 verifica se chegou ao final da expressão (Token seguinte nulo ou não operador). Caso sim, o conteúdo armazenado em P1 será convertido em comandos da máquina alvo e escrito no *buffer* de saída. A título de curiosidade, pode ser visto na Listagem 1 o microcódigo correspondente ao estado V_2.

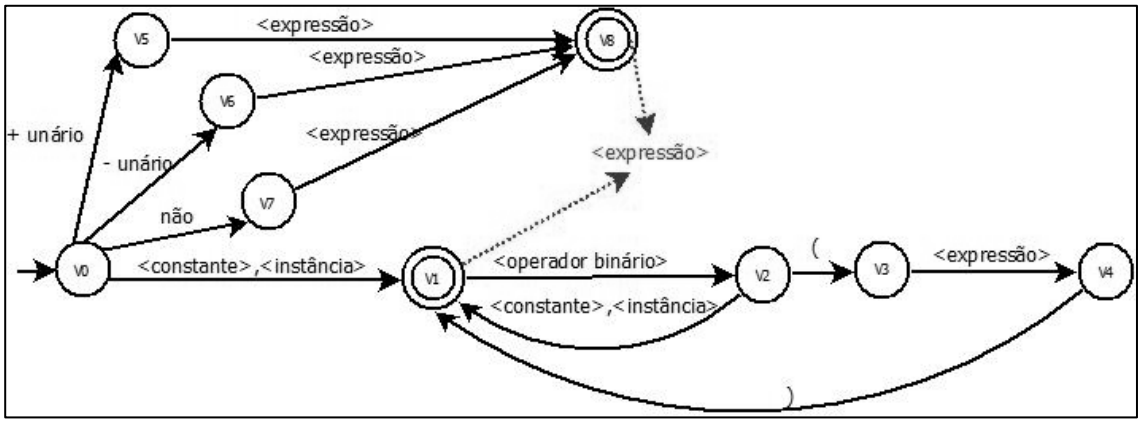

Figura 2. Parte da MM que reconhece expressões (fonte: do Autor)

O autômato pode ser percorrido de forma recursiva. Cada estado possui um atributo chamado código de saída. Valores negativos indicam erro no processamento da sentença. Valores positivos indicam o tipo de sentença encontrada. Esse tipo de sentença são tokens especiais (os códigos entre 51 e 100 citados anteriormente). Somente estados

finais possuem valores nesta faixa (na Figura 2 os códigos de saída estão representados por linhas pontilhadas indicando o tipo de sentença – no caso: <expressão>).

Listagem 1. Exemplo de microcódigo do estado V_2 (fonte: do Autor)

```
wh1:                        ;rótulo
jmp_pr_tk_mn_tp1 endwh1     ;salta se prioridade(token)<prioridade(topo de P1)
p1deqtoreg                  ;desempilha topo de P1 para registrador
p2endreg                    ;empilha registrador em P2
jmp wh1                     ;salta para "wh1"
endwh1:                     ;rótulo endwh1
p1enqtk                     ;empilha token em P1
```

Os estados V_1 e V_8 retornam como código de saída o código 51, que equivale à sentença do tipo <expressão>. O código de saída igual à zero é reservado para o estado inicial. Uma aresta pode ser acionada por esse código de sentença. Quando não for encontrada uma aresta de saída para o token lido na fita, o algoritmo armazena na pilha P_3 o estado atual e salta para o estado inicial, continuando na posição corrente da fita. Ao chegar a um estado final e não houver mais como fazer transição, o algoritmo guarda o código de saída do último estado, retorna ao estado anteriormente armazenado em P_3 e verifica se nesse estado existe uma aresta acionada pelo código de saída do estado anterior. Se houver uma aresta válida é feita a transição para o próximo estado. O motivo de utilizar a recursão é reaproveitar caminhos já definidos no grafo. A Figura 3 mostra como alguns comandos estão definidos na MM através do uso da recursão. Observe o leitor que o estado V_{19} possui uma aresta que espera que uma sentença do tipo <expressão> seja encontrada. O algoritmo põe o estado V_{19} na pilha e vai para o estado V_0. Se os tokens seguintes na fita formarem uma expressão lógico-aritmética válida, o último estado visitado será V_1 ou V_8 (ver Figura 2), cujo código de saída é <expressão>. Não sendo mais possível fazer transição, o algoritmo retorna para o estado contido no topo da pilha P3, isto é, v_{19} e procura por uma aresta que seja acionada pelo token <expressão>. No caso, a única aresta disponível do estado V_{19} será acionada e a transição será feita para o estado V_{20}. Ainda na Figura 3 podem ser vistas outras sentenças aceitas pela MM. A sentença "Diga <expressão>", por exemplo, instrui o algoritmo é exibir uma mensagem para o usuário. A Listagem 2 exibe um exemplo com

Listagem 2. Exemplo de sentença utilizando bloco (fonte: do Autor)

```
1:número contador <- 3
2:enquanto contador > 0 {
    diga "Contador é "+contador;
    contador <- contador - 1;
  }
```

duas sentenças informadas em sequência. A sentença constante na linha 1 cria uma variável global do tipo numérico chamada contador e atribui a ela o valor numérico 3. A segunda sentença faz uso de um laço de repetição do tipo enquanto. Ao ser passada ao computador, este exibirá as mensagens: "Contador 3", "Contador 2" e "Contador 1".

A linguagem reconhecida pela MM é uma LNC em português que possui estruturas semelhantes a uma linguagem de programação comum. Além da análise de expressões, ela também reconhece: expressões lógico-aritméticas, comandos, definição de funções, declaração de classes, variáveis, atribuições, estruturas condicionais, laços

de repetição; entre alguns outros comandos. Alguns comandos, como atribuição de valores a instâncias, realizam alteração na estrutura do autômato. No caso da atribuição, um novo caminho com arestas que definem o nome da instância é acrescido ao autômato. O que permite nomes de instâncias compostos por mais de uma palavra ("Nome da Pessoa" seria um identificador válido para uma instância, por exemplo). Obviamente, não podem ser declaradas novas instancias com caminhos já existentes e nem contendo operadores ou palavras reservadas (o código da máquina alvo que cria instâncias faz esta e outras verificações).

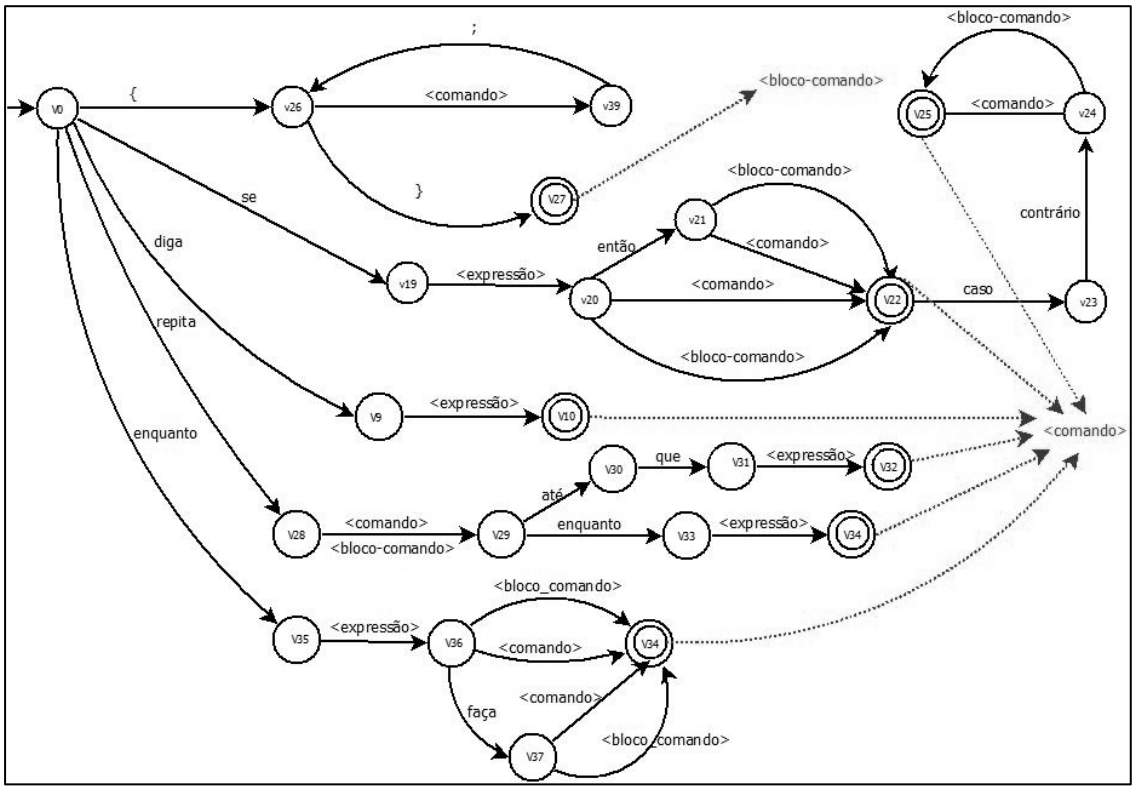

Figura 3. Parte da MM que reconhece alguns comandos e bloco de comandos (fonte: do Autor)

De forma semelhante à declaração de instâncias, uma função também pode ser declarada. Uma função é uma sequência de palavras e/ou parâmetros que levam a um estado final que gera um código pré-definido. A listagem 3 mostra duas funções sendo definidas. Na linha 1 é feita a declaração da função "toque música <texto arquivo>. O operador "<-" indica que aquela sequência de palavras irá acionar o comando ou bloco de comandos a seguir (apesar de o código ter sido omitido, considere o leitor que a máquina virtual do algoritmo possui comandos de alto nível que fazem chamadas a rotinas externas). Também é possível que dois ou mais caminhos diferentes possam levar a um mesmo estado final, permitindo que várias expressões diferentes sejam utilizadas para gerar o mesmo resultado – como acontece na linguagem natural. O algoritmo permite que diversas formas possam ser utilizadas para executar um mesmo comando, gerando uma espécie de polimorfismo de sentenças. É o que está acontecendo na linha 2 da Listagem 3, onde a função "quero ouvir música <texto arquivo>" está sendo definida a partir de uma função pré-existente. No caso, essa sentença instrui o algoritmo a criar uma chamada redundante para a função definida na linha 1. A capacidade de reconhecimento de novas sentenças aumenta na medida em que novos

dados são inseridos no autômato. As alterações geradas na MM podem ser vistas na Figura 4. O código definido a direita do sinal "<-" na linha 1 da figura 4 irá compor o estado v_{107}. Os parâmetros de funções são tratados em um blocos como variáveis locais.

Listagem 3. Exemplo de definição de funções (fonte: do Autor)

```
1:toque música <texto arquivo> <- {<código para tocar uma música>}
2:quero ouvir música <texto arquivo> <- toque música arquivo
```

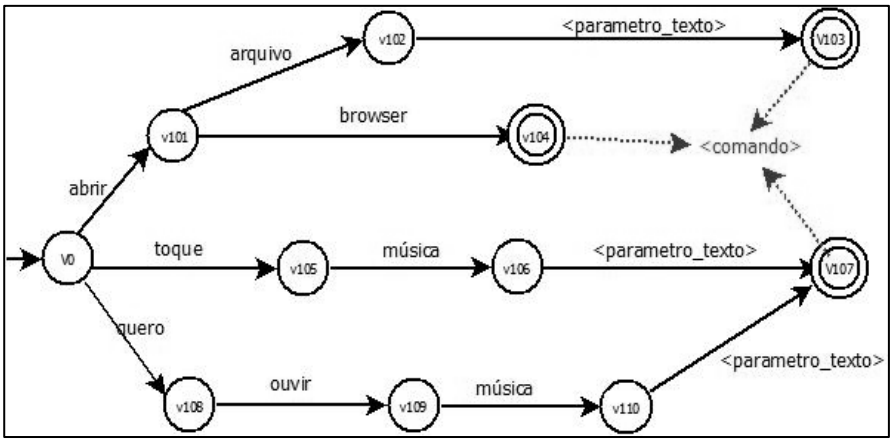

Figura 4. Alterações geradas na MM pelo código da Listagem 3 (fonte: do Autor)

Observe que as sentenças "Toque música <parâmetro_texto>" e "Quero ouvir música <parâmetro_texto>" realizam a mesma operação. Então, tanto faz o usuário usar uma sentença como a outra. O resultado será o mesmo. E a chamada feita pelo usuário será algo como "Toque música <minha música>" ou "Quero ouvir música <minha música>". Onde "<minha música>" é uma sentença que leva a um estado final que representa uma instância do tipo binário (arquivo mp3, por exemplo). Esses novos caminhos juntamente com a possibilidade de redundância culminam por definirem uma segunda linguagem baseada em frases e parâmetros, que é gerada a partir da linguagem núcleo. À medida que novas funções são cadastradas, novos caminhos são gerados no autômato e a linguagem núcleo original é expandida. Cabe salientar que a linguagem núcleo é imutável – ou seja – os vértices e arestas que a reconhecem não podem ser alterados. Porém os demais vértices, que correspondem à linguagem expandida, podem ser manipulados pelo usuário através da sintaxe da linguagem núcleo.

Um estado final pode também conter instruções que carreguem (LOAD) um determinado objeto na memória – como é o caso das instâncias. Abstraindo o código executável que faz a carga deste objeto, pode-se inferir que, além de um código executável, estados finais podem armazenar uma informação de qualquer natureza. Também é possível armazenar dados sobre a definição de uma classe, uma instância de uma classe, uma música, imagem ou qualquer outro objeto. No sistema existe uma tabela de símbolos, mas sua função é apenas para otimização do algoritmo. Todos os símbolos são definidos como vértices dentro do próprio autômato. Um estado final de definição de tipo conterá informações a respeito deste tipo. Além de apontadores para todos os estados finais que são instâncias suas. Tem-se então que existem arestas relacionando os estados finais. Da mesma sorte, os relacionamentos entre estados finais que representam definições de classes ficam armazenadas no grafo representado pelo

autômato. Essa configuração pode também ser interpretada como uma forma de representação de conhecimento – similar a uma ontologia. De maneira que a MM aqui proposta, além de processar a linguagem núcleo e expandi-la, pode ainda ser considerada como uma forma de representação do conhecimento.

A máquina virtual é bastante simples. Basicamente é um conjunto de quatro *threads* que acessam uma fila de prioridade contendo *buffers* de saída formada a cada sentença passada pelo do usuário. Cada sentença gera um processo, com sua própria pilha e área de memória. A cada segundo é feita a troca de contexto, caso mais de quatro processos estejam sendo executado por vez.

Cabe salientar que os microcódigos são de uso interno do sistema. Quando novos caminhos forem criados – e o serão através da linguagem núcleo – o usuário não precisará definir nenhum microcódigo. O próprio sistema o fará de forma implícita. O usuário final apenas deverá saber usar a sintaxe da linguagem núcleo que permita essa criação. Em termos de implementação, a Máquina de Moore é um grafo armazenado em disco através de uma base de dados desenvolvida especificamente para este fim. Os vértices necessários para analisar determinada sentença são carregados na memória. Vértices não utilizados após um número parametrizável de ciclos da máquina virtual são salvos em disco e removidos da memória. Cada componente da Máquina de Moore (vértices e arestas) é vinculado a um pacote que pode ser exportado ou importado. Pode-se afirmar que desenvolver um pacote em um computador e exportá-lo para outra máquina é o mesmo que transferir conhecimento de uma base de dados para outra.

4. Considerações Finais

Este trabalho abordou a possibilidade de analisar sentenças informadas pelo usuário fazendo uso de uma Máquina de Moore para traduzir uma sentença passada pelo usuário em ações executadas pelo computador. Apesar de ainda não está completamente definido, já é possível ver alguma vantagem sobre os *chatterbots* e assistentes tradicionais. Enquanto estes apenas analisam frases estáticas, o algoritmo proposto permite processar comandos que possuam alguma lógica de programação. E considerando que o objetivo não é conversação, mas reagir às sentenças passadas pelo usuário – seja com uma resposta ou executando algo útil; o algoritmo proposto tem mostrado um grande potencial.

O algoritmo ainda está em fase de implementação e testes. A linguagem escolhida para desenvolvimento foi Java, utilizando o *toolkit GUI Swing*. A Máquina de Moore ainda não está completamente definida. Contudo, o que foi mostrado neste artigo já está implementado. A exceção da exportação de pacotes, que por questões de controle e segurança está sendo modificada. Será criado um repositório de pacotes na internet onde serão armazenados os pacotes. A época de submissão deste artigo, este servidor ainda não estava funcionando. Mas foi registrado o domínio www.meuajudantevirtual.com.br, onde futuramente estará disponível o software para download bem como pacotes, exemplos de uso e outras informações sobre o algoritmo.

Como os vértices são salvos em disco e carregados à medida que são solicitados, a performance é um pouco lenta. Pesquisas têm sido feitas com o intuito de melhorar a eficiência do algoritmo. Falta ainda concluir o reconhecimento de classes e alguns ajustes no tratamento do escopo de variáveis locais em blocos de comandos e declaração

de funções. Por enquanto, apenas tipos primitivos são reconhecidos em expressões. Estão previstas ainda inserir na linguagem núcleo algumas outras estruturas não comentadas aqui, tais como:

- Quando <expressão> faça <bloco_comando> (disparado quando a expressão for verdadeira);
- Para cada <elemento de conjunto, lista etc.> faça <bloco_comando>;
- Estrutura caso;
- Reconhecimento de conjuntos;

Para impedir a instabilidade do algoritmo, algumas limitações tiveram que ser impostas. Por exemplo: não é possível criar identificadores começando com constante de qualquer tipo. Palavras reservadas (se, enquanto, repita, entre outras) e operadores não podem ser utilizados em nome de funções ou variáveis. Restrições essas típicas de qualquer linguagem de programação. Para o usuário comum, utilizar o software é bastante simples, basta digitar uma frase ou pergunta e o sistema fornece a resposta conforme o conhecimento armazenado no MM. Passar o conhecimento para a base de dados do software já é um pouco mais complexo, pois implica no conhecimento da sintaxe da linguagem núcleo e lógica de programação. Pesquisas têm sido feitas visando à possibilidade de agregar ao algoritmo uma forma de aprendizado automático ou semiautomático, usando uma tabela de sinônimos para expandir a linguagem.

A fim de se avaliar o desempenho do algoritmo e como parte da pesquisa a ser desenvolvida, será realizado um estudo de caso onde serão criados dois pacotes: o primeiro será um pacote com para análise de sentenças genéricas típicas de *chatterbots* tradicionais. Tais como "Qual o seu nome?", "Que horas são?", por exemplo; e um segundo pacote contendo informações turísticas de uma cidade brasileira. Foi escolhida a cidade de Natal – RN. O resultado esperado é que o software seja capaz de responder a perguntas em linguagem natural tanto no contexto genérico quanto no contexto específico. A habilidade de reconhecer e responder as sentenças informadas será comparada aos resultados obtidos em outros softwares do gênero. Nesse teste será também verificada a escalabilidade do software.

Por fim, a capacidade de interpretar e reagir corretamente a diversas sentenças informadas cresce na medida em que novos caminhos e conhecimentos são inseridos na base de dados. O que implica em dizer que a capacidade do algoritmo tende a crescer com o tempo de uso.

5. Referencias

Aho, A. V., Sethi R., Ullman J. D. (1995) Compiladores: Princípios, técnicas e ferramentas. Rio de Janeiro - RJ. LTC Editora.

Babini, M. (2006) Reconhecimento de Padrões Lexicais por meio de Redes Neurais. Ilha Solteira – SP: Universidade Estadual Paulista. Faculdade de Engenharia de Ilha. Dissertação de Mestrado.

Bedregal B. R. C., Acióly, B. M. e Lyra, A. (2010) Introdução à Teoria da Computação: Linguagens Formais, Autômatos e Computabilidade. Natal. Edições UNP/FAPERN.

Comarella, R. L.; Café, L. M. A. (2008) Chatterbot: conceito, características, tipologia e construção. Informação & Sociedade: Estudos, João Pessoa, v.18, n. 2, p.55-67, maio/ago. 2008.

Gruber, T. R. (1993) A translation approach to portable ontology specifications. Stanford: Knowledge Systems Laboratory. 27p. Technical Report KSL 92-71.

Jager, G.; Rogers, J. Formal language theory: refining the Chomsky Hierarchy. Philosophical Transactions of the Royal Society of London B: Biological Sciences, The Royal Society, v. 367, n. 1598, p. 1956-1970, 2012. ISSN 0962-8436. Disponível em: <http://rstb.royalsocietypublishing.org/content/367/1598/1956>.

Kornai, A. Natural languages and the Chomsky Hierarchy. In: Proceedings of the Second Conference on European Chapter of the Association for Computational Linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics, 1985. (EACL 85), p. 1$^{\perp}$ 7. Disponível em: <https://doi.org/10.3115/976931.976932>.

Kuhn, T. e Schwitter, R. (2008) Writing Support for Controlled Natural Languages. In: Powers, D., Stockes, N. (eds.) Proceedings of ALTA, pp. 46-54.

Lancaster, F. W. (2003) Indexação e resumos: teoria e prática. Brasília. Briquet de Lemos.

Laven, S. The Simon Laven Page. Disponível em: <http://www.simonlaven.com>. Acesso em: 26/11/2016.

Menezes, P. B. (2011) Linguagens Formais e Autômatos. 6ª ed. Porto Alegre: Artmed.

Neto, J. M. de O., Tonin, S. D.; Prietch, S. (2010) Processamento da Linguagem Natural e suas aplicações computacionais. Escola Regional de Informática (ERIN).

Neves, A. M. M., Barros, F. A. (2005) iAIML: Um Mecanismo para Tratamento de Intenção em Chatterbots. In: Congresso da Sociedade Brasileira de Computação. 25., São Leopoldo. Anais. São Leopoldo. p.1032-104.

Primo, A. e Coelho, L. R. (2002) Comunicação e inteligência artificial: interagindo com o robô de conversação Cybelle. In: MOTTA, L. G. M. et al. (Eds.). Estratégias e culturas da comunicação ed. Brasília. Brasília: Editora Universidade de Brasília. p. 83-106.

Schneider, M. O. (2001) Processamento de Linguagem Natural (PLN). Campinas: Pontifícia Universidade Católica de Campinas - Curso de Mestrado em Sistemas de Computação. 28p. Apostila de Processamento de Linguagem Natural.

Vieira, J. N. (2006) Introdução aos Fundamentos da Computação – Linguagens e Máquinas. Thomson Learning.