Implementação de Compilador e Ambiente de Programação Icônica para a Linguagem Logo em um Ambiente de Robótica Pedagógica de Baixo Custo

Marcelo R. G. Barbosa[1], Felipe A. Silva[2], Victor M. de A. Oliveira[2], Valéria D. Feltrim[1], Luiz G. B. Mirisola[2], Paulo C. Gonçalves[1], Josué J. G. Ramos[2], Lucas T. Alves[2]

[1] Departamento de Informática – Universidade Estadual de Maringá – UEM
Maringá – PR – Brasil

[2] Centro de Tecnologia da Informação Renato Archer – CTI
Campinas – SP – Brasil

marcelo.barbosa@benner.com.br, {fel1310, victormatheus, luiz.mirisola, paulocg, ltanure}@gmail.com, valeria.feltrim@din.uem.br, josue.ramos@cti.gov.br

Abstract. *Pedagogical Robotics is a tool to promote the integration of practical activities with scientific knowledge and team work that can become feasible in developing countries by using low cost alternatives. In this sense the availability of hardware and software components will contribute to the development of pedagogical robotics programs. This paper presents a contribution to this objective by presenting a Logo Compiler that use the formalisms required by compilers and the process of Logo code generation in an Iconic Logo programming environment*

Resumo. *A robótica pedagógica constitui um instrumento para promover a integração de atividades práticas com o conhecimento cientifico e trabalho em equipe, que pode ser viabilizado em países em desenvolvimento com alternativas de baixo custo. Neste contexto, a disponibilização de componentes de hardware e software abertos vem a favorecer o desenvolvimento de programas de robótica pedagógica. Este artigo apresenta mais uma contribuição neste sentido, pelo desenvolvimento de compilador Logo aberto que usa os formalismos requeridos pela técnica de desenvolvimento de compiladores e pela apresentação do processo de geração de código Logo a partir de um ambiente icônico de programação.*

1. Introdução

Ao longo dos últimos anos, as ideias relacionadas ao *construcionismo* de Papert [Papert 2004] têm motivado o desenvolvimento de diferentes sistemas computadorizados voltados para a educação. Nesse contexto se destaca a robótica educacional, unindo o uso da linguagem Logo, proposta por Papert, à montagem e programação de robôs a partir de *kits* de robótica especialmente preparados para esse fim. O conjunto de robótica educacional LEGO *Mindstorms*, comercializado pela empresa LEGO, foi um dos primeiros *kits* disponíveis no mercado com essas características [Gonçalves 2007].

Com a popularização de ambientes robóticos educacionais, diversas iniciativas surgiram em nível mundial. No âmbito brasileiro, há vários ambientes fornecidos por empresas brasileiras e estrangeiras, além de ambientes abertos como o *Arduino* [Arduino 2009] e *GoGo board* [Sipitakiat 2004]. Esses ambientes possuem uma unidade de controle básica e um padrão próprio de adição de componentes, como sensores e motores. Entretanto, conforme detalhado em Ramos *et al.* (2007), o custo de tais ambientes constitui o principal obstáculo para o seu uso de forma difundida nas escolas, especialmente em países em desenvolvimento. Dessa forma, a disponibilização de soluções abertas de baixo custo constitui uma alternativa importante para viabilizar o uso da robótica educacional de forma abrangente.

Além do custo, outras questões interferem na aplicação de ambientes como o disponibilizado para a *GoGo board* em plataformas de baixo custo, por exemplo, o software utilizado para a programação da placa. Ramos *et al.* (2009) propõe soluções de software que resolvem parte das limitações da *GoGo board*, visando sua aplicação no contexto educacional e usando computadores de baixo custo. As principais limitações abordadas são: i) possuir somente ambiente de programação textual (não possui ambiente de programação icônico); ii) a unidade de controle que opera com interface serial, dificultando o seu uso em computadores que possuem apenas interface USB, visto problemas de compatibilidade com vários conversores USB-serial e iii) ter software não compatível com o sistema operacional Linux. Além disso, o compilador utilizado pelo ambiente da *GoGo board* possuía código fechado, apesar dos esquemas de hardware e o código fonte da unidade de controle estarem abertos.

Dessa forma, as soluções propostas em Ramos *et al.* (2009) incluem: ambiente de programação textual e icônico, ambiente de operação e compilador, desenvolvidos como software aberto, além da possibilidade do uso do *GoGo board* em Linux. Também foi apresentada a adequação realizada no software e hardware da *GoGo board* para que esta seja operada segundo interface USB. Estes aspectos preenchem algumas lacunas que impediam a implementação de programas de Robótica Pedagógica de Baixo Custo (RPBC) em escolas e dificultavam a evolução de funcionalidades, como o uso de interfaces diferentes da serial convencional RS232.

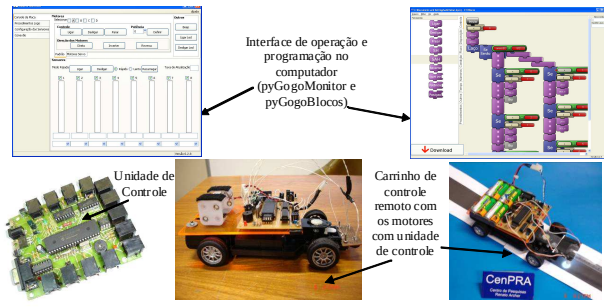


Figura 1(a). Componentes de ambiente RPBC, incluído interfaces do software de programação e operação, hardware da unidade controle, a *GoGo board* e experimento com carrinho seguidor de trilha.

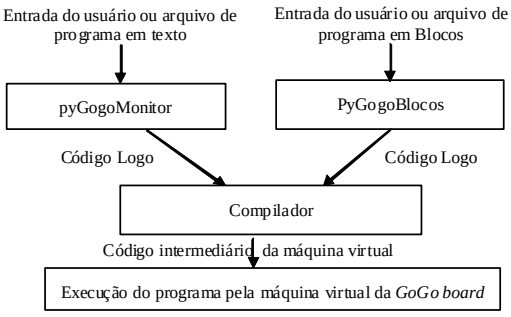


Figura 1(b). Integração dos componentes de software com a *GoGo board*.

A Figura 1(a) apresenta os componentes de RPBC: na parte superior estão as interfaces do software de programação e operação; na parte inferior, o hardware da unidade de controle (a *GoGo board*) e a aplicação desta em um experimento de carrinho seguidor de trilha. A Figura 1(b) apresenta como os componentes de hardware e software interagem. Nesta figura, é mostrado o ambiente para operação e programação da placa, chamado pyGoGoMonitor, que tem recursos para operação remota da unidade de controle e para o desenvolvimento de programas textuais na linguagem Logo. Ao lado é mostrado o ambiente de programação icônico, chamado pyGoGoBlocos, utilizado para se gerar programas Logo de forma gráfica. Ambos se comunicam com a *GoGo board* utilizando um compilador, que opera no computador de controle, traduzindo programas em Logo para a linguagem utilizada pela *GoGo board* para, em seguida, enviar estes programas para serem carregados e armazenados na *GoGo board*. O usuário então pode comandar a execução do programa armazenado para realizar uma aplicação robótica.

O compilador desenvolvido, isto é, o software que traduz os comandos da linguagem Logo para a linguagem interpretada pela a unidade de controle utiliza ferramentas especificas de desenvolvimento de compiladores baseados em uma especificação formal da linguagem, o que facilita a manutenção do software e a eventual incorporação de novos comandos. Como não existe uma especificação formal universalmente reconhecida da linguagem Logo, optou-se por adotar como padrão à especificação utilizada no compilador com código fechado desenvolvido originalmente para a *GoGo board* [Sipitakiat 2004].

Assim, este trabalho apresenta os aspectos ligados à implementação do compilador utilizado pelos ambientes pyGoGoMonitor e pyGoGoBlocos, que foi inicialmente desenvolvido na UEM [Barbosa 2008] e finalizado no CTI. No estado atual, o compilador permite a definição de comando de motores, leitura de sensores, controle de fluxo, variáveis e constantes numéricas, operações numéricas e lógicas, temporização e procedimentos. Também é mostrado neste trabalho o processo de geração de código Logo no ambiente programação icônica.

Após esta seção introdutória, o restante deste artigo está organizado como segue. A Seção 2 apresenta a máquina virtual Logo da unidade de controle. A Seção 3 apresenta o compilador desenvolvido. A Seção 4 apresenta o ambiente de programação gráfico e, finalmente, a Seção 5 apresenta as conclusões deste trabalho.

2. A Máquina Virtual Logo da Unidade de Controle

Desde a sua criação, por volta de 2002, o desenvolvedor da *GoGo board* distribuiu os esquemas eletrônicos, guias de montagem, guias de experimentos, etc., visando disponibilizar uma unidade controle adequada para experimentos com robótica de baixo custo. Recentemente, em 2007, foi distribuído sob licença GPL o software da unidade de controle da *GoGo board*, entretanto, não foram disponibilizados os softwares ligados à unidade de programação e operação.

Entre os componentes de software disponibilizados está a máquina virtual que interpreta os códigos de instrução segundo uma linguagem intermediária específica, em geral, chamada de *bytecode*. Uma máquina virtual corresponde a um processador virtual que tem um conjunto de instruções com nível de abstração mais baixo que a linguagem

em que se está escrevendo um programa, porém com um nível de abstração mais alto que a linguagem do processador. Dessa forma, um compilador que produz código para uma máquina virtual não depende do processador utilizado na unidade de controle.

Assim, são pré-carregados na unidade de controle funções responsáveis pela execução das instruções da máquina virtual correspondente a tradução de um programa na linguagem Logo escrito pelo usuário. Este programa é carregado após a compilação e, na medida em que o usuário comande a sua execução, o conjunto de instruções compilado é executado, permitindo a execução autônoma de programas escritos nesta linguagem, compondo, assim, um robô autônomo de baixo custo.

No caso de uma *GoGo board* com interface serial, que tem velocidade de comunicação pequena, é justificável o uso de uma máquina virtual, pois isso diminui o tempo de carregamento de programas, já que o tamanho dos mesmos fica menor em relação à opção de carregamento de todas as funções que seriam requeridas por uma determinada aplicação. No caso de placas com interface USB, com comunicação quase dez mil vezes mais rápida do que a comunicação serial RS232, esta alternativa pode ser revista, considerando-se a tradução direta dos comandos em Logo para os códigos próprios do processador.

Dentre as diferentes alternativas para a máquina virtual, os desenvolvedores da *GoGo board* implementaram uma máquina virtual em pilha, de modo que os valores a serem processados são empilhados para posterior execução da operação, sendo que o resultado normalmente fica armazenado no topo da pilha de dados. Em geral, uma instrução de um programa Logo corresponde a várias instruções da máquina virtual. Por exemplo, se o objetivo é adicionar dois números, a tradução do comando SOMA é feita para uma linguagem intermediaria semelhante a: coloque o primeiro valor na pilha, então coloque o segundo valor na pilha e some os dois números, colocando o resultado na pilha.

Em 2007, Arnan Sipiatkiat disponibilizou o código-fonte que mostra como o interpretador de *bytecode* da linguagem Cricket Logo [Cricket 2009] interpreta cada código. Este interpretador é chamado de Cricket Logo Virtual Machine (VM). Neste código-fonte estão descritos os códigos de máquina aceitos pela *GoGo board*, além de outras características do interpretador, como a notação pós-fixa, que define como uma expressão se comportará ao ser analisada.

Como a Critcket Logo VM tem uma arquitetura simples, a tradução do código-fonte para o código intermediário corresponde a uma sequencia de *bytecodes*. Logo, a tradução do código Logo "*Repeat 10 [onfor 20 beep]*", corresponde ao envio dos *bytecodes* "1 10 3 5 1 20 50 12 4 9 0". Este comando *Repeat* define a uma lista de valores que deverão ser repetidos 10 vezes. Seguindo a forma pós-fixa, será empilhado o valor 10 (*bytecode* 10), que corresponde a um byte (*bytecode* 1) – empilha "1 10"; em seguida será empilhada a lista de valores (*bytecode* 3) com o seu tamanho (*bytecode* 5). Este tamanho corresponde aos *bytecodes* enviados a partir do tamanho da lista (no exemplo, os *bytecodes* 1 20 50 12), até o seu final (*bytecode* 4), inclusive, seguida do comando de iteração *Repeat* (*bytecode* 9) – empilha "3 5 1 20 50 12 4 9". Por fim, é empilhado o marcador de término do código (*bytecode* 0).

3. O Compilador

Um compilador traduz um programa descrito em uma linguagem de alto nível, mais adequada aos seres humanos, para os códigos em uma linguagem de máquina, que podem ser executados por um processador. No presente caso, a linguagem Logo é a linguagem de alto nível e os *bytecodes* interpretados pela máquina virtual da *GoGo board* é a linguagem de máquina. Dessa forma, o compilador apresentado aqui traduz instruções Logo para instruções da Cricket Logo VM. Grosso modo, o processo de compilação pode ser descrito como contendo as seguintes etapas: análise léxica, análise sintática, análise semântica e geração de código.

Para o desenvolvimento do compilador foi utilizada a ferramenta Ply [Beazley 2001]. Suas principais características são: estar implementada em Python; ser similar às ferramentas tradicionais Lex/Yacc do Unix; prover suporte às regras de recuperação e checagem de erros, gramáticas ambíguas e produções vazias; e ser relativamente simples de usar. A Ply possui dois módulos dedicados a geração automática de compiladores, o Lex.py e o Yacc.py, que serão descritos nas seções seguintes.

3.1. Análise Léxica

A primeira fase da compilação é a análise léxica. O seu objetivo é separar a seqüência de caracteres do texto de um programa-fonte em itens léxicos ou lexemas, que são sequencias de caracteres com um significado coletivo. Os lexemas são classificados como identificadores, palavras-chave, operadores, constantes, símbolos de pontuação, entre outros [Aho *et al.* 2008]. O analisador léxico realiza também outras tarefas, como remover comentários e marcas de edição (tabulações, caracteres de avanço de linha e espaços), bem como relacionar o número de linha com possíveis mensagens de erro.

Para a implementação do analisador léxico para a linguagem Logo foi utilizado o módulo Lex.py da ferramenta Ply. A Tabela 1 apresenta alguns lexemas da linguagem Logo, a sua classificação e exemplos simplificados da sua especificação no módulo Lex.

Tabela 1. Exemplos de lexemas da linguagem Logo

Lexemas	Classificação	Exemplo de definição Lex.py

to, end, repeat, if, ifelse, beep, stop, ledon, ledoff…	Palavras reservadas	`reserved = {'to' : 'TO',`
		`'end': 'END',`
		`… etc … }`

>, <, =, +, -, *, /, %	Operações e Relação	`t_DIVIDE = '/'`

Inicializa_, teste_123_, :tx, :valor, “num,“rec, ab, , c, ...	Identificadores	`digit='([0-9])'`
		`letter='([a-zA-Z])'`
		`alphanum='([a-zA-Z0-9])'`
		`Procname='(letter+(alphanum)*)'`

0, 13, 9, 168, ...	Números inteiros	`Nliteral='(digit)+'`
; (linha de comentário)	Comentários	`t_ignore_COMMENTLINE=';.*'`

Conforme mostrado nos exemplos da Tabela 1, os lexemas da linguagem Logo são especificados como regras definidas nos arquivos de entrada do módulo Lex no formato apropriado. O dicionário *reserved* é utilizado para definir as palavras reservadas da linguagem, isto é, lexemas simples, sem regras especiais de formação. Regras também podem ser definidas a partir de expressões regulares. Por exemplo, nomes de procedimentos são definidos como uma letra concatenada com uma seqüência de zero ou mais caracteres alfanuméricos (operador *). Uma constante numérica é definida

como uma seqüência de um ou mais dígitos (operador +). Funções podem ser associadas a lexemas, por exemplo, quando uma constante numérica é encontrada uma função verifica se o número é válido e emite uma mensagem de erro, caso necessário.

3.2. Análise Sintática, Semântica e Geração de Código

A seqüência de lexemas identificados pela análise léxica é a entrada do módulo Yacc.py. Este módulo foi utilizado para a implementação das análises sintática e semântica e também para a geração de código. Desta forma, o compilador desenvolvido se caracteriza como sendo de uma passagem [Louden 2004]. A análise sintática verifica se os lexemas formam um programa estruturado de acordo com a gramática da linguagem, que define os comandos e operações possíveis no Logo. A análise semântica verifica a correção semântica das instruções e é implementada por meio de regras que são disparadas durante a análise sintática. De modo similar, regras de geração de código são associadas à estrutura sintática da linguagem. O código gerado implementa as operações especificadas pelo programa na linguagem de saída do compilador. O Yacc.py requer a especificação de declarações (variáveis, precedências entre operadores) e regras de tradução (produção gramatical e da ação semântica), conforme exemplos mostrados na Tabela 2.

Tabela 2. Exemplos de regras de tradução para a gramática da linguagem Logo

Declarações	`DEBUG = True` `precedence = (('left', 'AND', 'OR', 'XOR'),.. }`

Regras de tradução	`'procedure : TO PROCEDURENAME statements END'`
	`'statement : IF expression LBRACKET statements RBRACKET'`
	`'statement : SHOW expression'`

As regras de tradução da Tabela 2 mostram que um procedimento (*procedure*) deve ser definido utilizando-se o lexema TO, seguido do nome do procedimento, um *statement* e finalizado pelo lexema END. Um *statement* é definido por outras regras da gramática, como as duas regras seguintes mostradas na tabela, que definem a estrutura condicional IF e o comando SHOW (que envia um valor pela porta serial).

A seqüência de lexemas é lida e processada da esquerda para a direita, aplicando-se as regras quando possível, até que todo o código seja reduzido a uma regra principal, que define que o programa deve ser uma seqüência de um ou mais procedimentos. Cada regra gramatical está associada a uma função, que gera o código correspondente à operação desejada, processa as regras filhas recursivamente e insere o código gerado na posição correta.

A linguagem inclui comandos especiais para operação do hardware (motores, sensores e beeps) e, caso hardware adicional seja desenvolvido, as regras léxicas e sintáticas podem ser facilmente estendidas para incorporar novos comandos à linguagem.

4. O Ambiente de Programação Gráfico - O pyGoGoBlocos

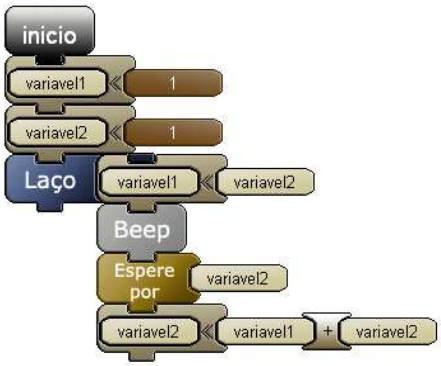
Linguagens de programação gráfica (ou visual) são aquelas em que o usuário especifica seus programas manipulando elementos visuais. As características que tornam estas linguagens atraentes, principalmente para as pessoas sem conhecimento de programação, são a capacidade de abstrair aspectos mais profundos de implementação e

permitir um foco maior no algoritmo que no conhecimento da sintaxe de uma determinada linguagem.

O pyGoGoBlocos está inserido nos dialetos da linguagem Logo com recursos gráficos. O desenvolvimento do pyGoGoBlocos foi motivado pela inexistência de uma plataforma com software aberto para programação gráfica para *kits* robóticos, conforme mostrado em Ramos *et al.* (2009). Assim, o pyGoGoBlocos, baseado no LogoBlocks [Begel 1996], usa uma interface de programação gráfica para ajudar usuários inexperientes a compreender os conceitos envolvidos na programação e na robótica. O encaixe dos blocos facilita a organização do pensamento e evita erros de sintaxe, muito comuns durante a aprendizagem.

A tarefa básica no pyGoGoBlocos é juntar pedaços de código em forma de blocos, de modo a criar o programa desejado. Cada componente da linguagem Logo é mapeado em um bloco, que possui uma determinada cor e tipos de conexões. As conexões são espaços onde se podem conectar outros blocos. Por meio da cor, passa-se a informação de tipo do bloco, como "condicional" ou "número". Por meio das conexões, visualiza-se claramente a sintaxe da linguagem, já que o formato de encaixe só permite que tipos compatíveis se conectem. Por exemplo, o bloco "SE" só aceita expressões booleanas na comparação e somente estas possuem a borda arredondada que se encaixa no espaço de comparação. Além disso, é possível encapsular código em procedimentos, minimizando a confusão de muitos elementos visuais na tela.

O pyGoGoBlocos faz o mapeamento dos esquemas de blocos em código textual, que é compilado e enviado à placa para ser executado. Na Figura 2 é mostrado um exemplo de programa para geração de sons a partir da sequencia de Fibonacci. À esquerda da figura, é mostrada a representação em blocos e, à direita, o código Logo correspondente.




```
global variavel1
global variavel2
to start
setvariavel1 1
setvariavel2 1
loop [
    setvariavel1 varia
    beep
    wait variavel2
    setvariavel2
variavel1+variavel2
]
end
```

Figura 2. Esquema de blocos (à esquerda) e o código Logo correspondente (à direita) que emite um som numa freqüência igual à seqüência de Fibonacci.

Os tipos de blocos disponíveis no pyGoGoBlocos podem ser classificados em oito categorias. A Tabela 3 apresenta as categorias de blocos disponíveis, bem como uma descrição dos blocos que compõem cada categoria.

Tabela 3. Categorias e tipos de blocos disponíveis no pyGoGoBlocos

Categoria	Tipo de Blocos
Controle	Ligar, desligar, frear, girar motor em sentido horário, girar motor em sentido anti-horário, reverter sentido, ligar por uma duração de tempo, alterar potência dos motores, alterar posição de motor servo.
Disposição	Espaço vertical, espaço horizontal, espaço vertical maior.
Fluxo	Se, se-senão, laço infinito, repetir um número de vezes, esperar até que uma condição seja verdadeira, parar a execução do código.
Condição	Comparação entre dois números (menor, maior, igual), e, ou, ou exclusivo, não, botão
Números	Número inteiro, número aleatório, pegar valor de um sensor, criar variável, atribuir valor a uma variável, soma, subtração, divisão, módulo. Obs.: onde se pode utilizar números, também se pode usar variáveis, que são mapeadas em tempo de execução pelo interpretador nos seus valores numéricos.
Temporização	Duração de tempo (décimos de segundo), esperar uma quantidade de tempo, pegar valor do contador de tempo, reiniciar contador de tempo.
Outros	Bip, acender led, apagar led, comentário.
Procedimento	Criar novo procedimento, finalizar procedimento.

4.1. Mecanismo para a Transformação dos Blocos em Código Logo

A implementação do pyGoGoBlocos usa programação orientada a objetos. O sistema de blocos está baseado na definição de uma classe "Bloco" que contem as propriedades gerais e cada bloco é uma especialização desta classe.

Assim, o algoritmo de geração de código a partir dos blocos baseia-se no fato que, por construção, cada bloco possui um conjunto de conexões e cada conexão possui um retângulo que quando entra em colisão com o retângulo de outra conexão verifica a possibilidade de encaixe considerando as propriedades de cada conexão. As conexões possuem as seguintes propriedades de i) Tipo: normal, lógica e número; ii) Formato: macho e fêmea e iii) Fluxo: para o pai, para o filho. O encaixe se dá em função das propriedades das duas conexões envolvidas e segue as seguintes regras: i) As conexões devem ter o mesmo tipo e ii) As conexões devem ter formatos opostos.

Os blocos e suas conexões formam uma árvore sintática. O processo de geração de código percorre esta árvore seguindo as conexões de cada bloco. Cada bloco declara como será gerado seu próprio código. Por exemplo, o bloco Beep está implementado em código Python pelas seguintes instruções:

```
self.add_code(self.conn_parent)
self.add_code("beep\n")
self.add_code(self.conn_child)
```

onde *self.conn_parent* é a conexão que liga o bloco com o bloco de cima e *self.conn_child* o liga com o bloco de baixo.

É utilizada a propriedade de poliformismo da orientação a objeto na função *add_code* para permitir que, quando o parâmetro da função for uma *string* (ex.: *"beep\n"*), este seja simplesmente inserido no código de saída. Quando o parâmetro é uma conexão, é seguida a conexão e, recursivamente, é analisado o código do bloco que está conectado a ela, até que o bloco verificado não possua conexões que possam ser seguidas. Neste ponto, um problema que pode surgir é haver uma recursão infinita, se uma conexão que se liga com o bloco "pai" for seguida. Por isso foi inserido a

propriedade Fluxo, permitindo que somente conexões do tipo “para o filho” sejam seguidas no processo de geração de código.

O pseudo-código do algoritmo de geração de código Logo a partir da estrutura de blocos é mostrado a seguir:

```
Código(bloco, texto):
   para todas as entradas E da função add_code:
      se E for uma conexão:
         se E estiver sendo utilizada e não apontar para o bloco pai:
            texto = Código( destino de E, texto )
      se E for um tipo constante (string, inteiro ou float):
         texto = texto + E
   retorne texto
```

A Figura 3 mostra a árvore que é percorrida durante o processo de geração de código para um programa em blocos. A partir do bloco início, busca-se o seu filho, que é o bloco Mudar variável, que está conectado à variável1 e ao valor 1, fazendo que seja gerado o código Logo para atribuir o valor 1 a variável1. Este bloco tem como filho outro bloco mudar variável que atribui valor 1 à variável dois. O filho seguinte é o bloco Laço, que está conectado a uma seqüência de blocos a ser executada repetidamente.

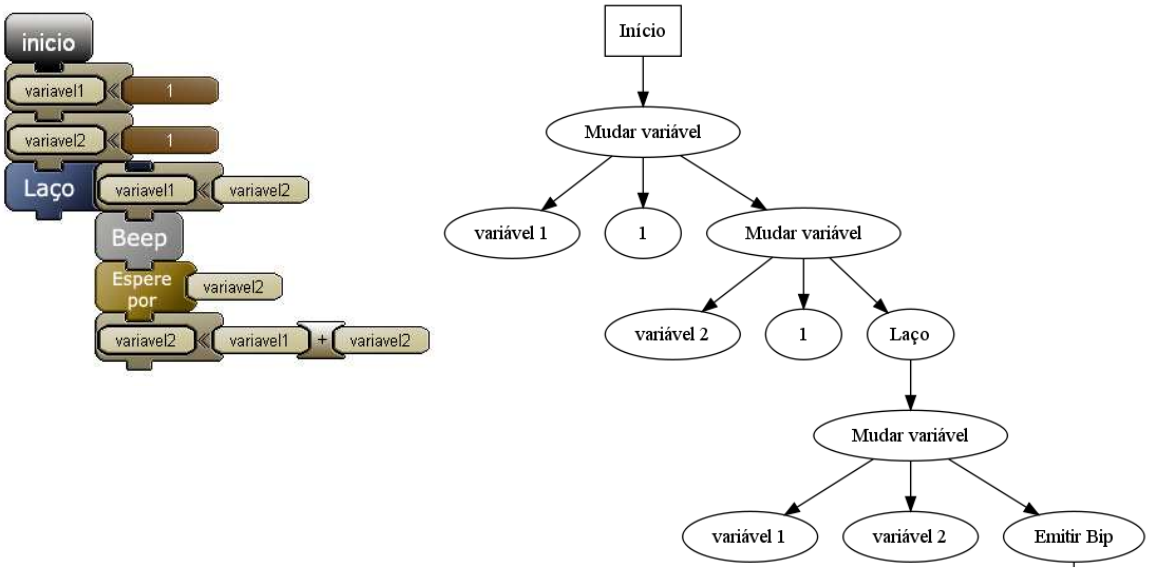


Figura 3. Mostra-se à direita, de forma parcial, o processo de geração de código para o programa em blocos mostrado à esquerda.

5. Disponibilidade do Projeto

O projeto descrito neste artigo está disponível sob licença GPL e pode ser encontrado no endereço <http://sourceforge.net/projects/br-gogo>.

6. Conclusão

Este artigo apresentou mais uma contribuição com objetivo de disponibilizar componentes de hardware e software abertos para o desenvolvimento de programas de robótica pedagógica de baixo custo. Foi apresentado um compilador Logo de código aberto, que usa os formalismos requeridos pela técnica de desenvolvimento de compiladores e o processo de geração de código Logo a partir de um ambiente icônico

de programação. Com os resultados apresentados neste artigo o sistema está pronto para inicio de testes em escolas.

Agradecimentos

Os autores agradecem o apoio do Programa PIBIC do CNPq.

Referências

Aho, A. V.; Lam, M. S.; Sethi, R.; Ullman, J. D. (2008) “Compiladores: Princípios, técnicas e ferramentas”. 2ª Edição, São Paulo: Pearson Addison-Wesley.

Arduino (2009) “Open-source physical computing platform”. Disponível em: <http://www.arduino.cc>. Acesso em: 30-07-2009.

Barbosa, M. R. G. (2008) “Estudo e Implementação de um Novo Compilador para a Linguagem de Programação Cricket Logo Visando sua Aplicação em uma Arquitetura de Baixo Custo”. Trabalho de Conclusão de Curso. Departamento de Informática, Centro de Tecnologia, Universidade Estadual de Maringá, PR.

Beazley, D. M. (2009) “PLY. Python Lex-Yacc”. Disponível em: <http://www.dabeaz.com/ply/>. Acesso em: 14-06-2009.

Begel, A. (1996) “A Graphical Programming Language for Interacting with the World”. < http://research.microsoft.com/~abegel/mit/begel-aup.pdf>. Acesso em: 17-10-2009.

Cricket (2009) “Handy Cricket Home Page”. Disponível em: <http://www.handyboard.com>. Acesso em: 30-07-2009.

Gonçalves, P. C. (2007) “Protótipo de um robô móvel de baixo custo para uso educacional” Dissertação (Mestrado em Ciência da Computação), Universidade Estadual de Maringá, Maringá, PR.

Louden, K. C. (2004) “Compiladores: Princípios e Práticas”. São Paulo: Editora Pioneira Thomson Learning.

Papert, S. (1994). “A Máquina das Crianças: Repensando a Escola na Era da Informática”. Porto Alegre: Artes Médicas.

Ramos, J. J. G.; Azevedo, H.; Vilhete V. A. J.; Noves O.; Figueiredo D.; Tanure L.; Holanda F. (2007) “Iniciativa Para Robótica Pedagógica Aberta e de Baixo Custo para Inclusão Social e Digital no Brasil”. In: Anais do VIII Simpósio Brasileiro de Automação Inteligente (SBAI 2007), Florianópolis, SC.

Ramos, J. J. G.; Silva, F. A.; V. Oliveira; Alves, L. T.; D’Abreu, J. V. V. (2009) “Desenvolvimento de Componentes de Hardware e software abertos para programas de Robótica Pedagógica de Baixo Custo”. In: Anais do IX Simpósio Brasileiro de Automação Inteligente (SBAI 2009), Brasília, DF.

Sipitakiat, A.; Blikstein, P.; Cavallo, D. (2004) “GoGo Board: Augmenting Programmable Bricks for Economically Challenged Audiences”, In: Proceedings of the Int. Conf. of the Learning Sciences (ICLS 2004), Los Angeles, USA.