VIII ERMAC-R3

5º Encontro Regional de Matemática Aplicada e Computacional
20 a 22-Novembro-2008
Universidade Federal do Rio Grande do Norte - Natal/RN

Transformando Funções Recursivas em Programas através de uma ferramenta gráfica de simulação

Alexandre Paes,
Mestrado em Modelagem Computacional do Conhecimento(MMCC)
Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Campus A. C. Simões BR 104 Norte Km 14 - 570,
E-mail: alexandre.paes.santos@gmail.com,

Eliana Almeida
Centro de Pesquisa em Matemática Computacional (CPMAT)
Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Campus A. C. Simões BR 104 Norte Km 14 - 570,
E-mail: eliana.almeida@gmail.com,

Resumo

O objetivo deste artigo é apresentar uma ferramenta de simulação que transforme funções recursivas em Programas numa linguagem de programação minimal. Com isso tenta-se prover um mecanismo de auxílio ao ensino de Computabilidade em Teoria da Computação.

Palavras-chave

Computabilidade, Funções Recursivas, Teoria da Computação.

Introdução

A tese de Church-Turing permite alinhar modelos de computação (Funções Recursivas, λ-Cálculo e Máquinas de Turing, por exemplo) com formalismos distintos, porém com um só objetivo, a concepção de um procedimento efetivo de cálculo, a saber, o Algoritmo. Efetivamente, toda ação de um computador representada por um algoritmo se resume a calcular o valor de uma função, pertencente à classe de funções recursivas, com determinados argumentos. Em [4] é chamada a atenção para o fato de que qualquer procedimento computacional para qualquer tentativa de formalizar a noção intuitiva de função computável pode ser traduzido em outra formalização, de tal forma que as duas formalizações têm as mesmas saídas para as mesmas entradas.

Entender a relação desses modelos computacionais é uma tarefa árdua para o aluno neófito, por ser exigido um alto grau de abstração, uma vez que são modelos fundamentados na Matemática. Acredita-se que tais dificuldades poderiam ser atenuadas com a construção de ferramentas que auxiliassem o discente no processo de aprendizagem. Este artigo apresenta uma ferramenta de fácil utilização que permite entender o conceito de funções computáveis, por meio do seu mapeamento em um programa escrito numa Linguagem Algorítmica Minimal (LAM), com o menor número de comandos necessários para escrever todos os Programas ditos computáveis.

A ferramenta aqui apresentada foi desenvolvida em JAVA e é parte integrante do projeto ACARAJÉ [11], que provê um conjunto integrado de ferramentas gráficas que simulam alguns dos modelos abstratos que constituem evidências para a Tese de Church-Turing.

Fundamentação Teórica

Serão aqui tratados os conceitos associados à classe das funções recursivas e à linguagem algorítmica, conforme [2].

Funções Recursivas

Ao elaborar sua prova, Gödel introduziu uma nova classe de funções que originalmente foram chamadas de funções recursivas, no entanto hoje essas mesmas funções são conhecidas como recursivas primitivas, nomenclatura dada pela primeira vez por Stephen Cole Kleene anos depois, em 1936. Kleene utilizou a numeração de Gödel para provar que não existe procedimento efetivo para decidir se tal artifício define um sistema de equações gerais como funções recursivas, construindo assim um formulação de uma teoria das funções recursivas. Outros matemáticos e lógicos já haviam percebido a existência dessas funções como Dedekind(1888), Hilbert(1926), Ackermann(1928) dentre [2]. As funções recursivas primitivas são uma classe especial de funções em geral e são muito importantes em Computabilidade na medida em que são mecânicas no sentido de mecânico que Gödel pela primeira vez tornou preciso. A recursividade permite que ao invés de se tratar de objetos infinitamente grandes, o matemático pode criar uma regra finita de construções. A teoria das funções recursivas é um tema central da Ciência da Computação. Essa classe de funções é construída a partir de outras funções elementares de uma forma composicional.

As Funções Recursivas Primitivas são construídas a partir das funções ditas iniciais ou básicas, onde seu efeito é conhecido, e as operações de composição, combinação e expoentização, que serão definidas a seguir. Ou seja, as Funções Recursivas Primitivas são geradas a partir da aplicação de um número finito de operações sobre as funções iniciais. Tais funções serão definidas sob o domínio dos números naturais, W, onde, para todo n $\geq$ 0, W^n corresponde a W x W x ... x W, n vezes. A seguir, serão listadas as funções elementares, incontestavelmente computáveis. Todas estas funções tratadas serão sempre entre números naturais, entidades matemáticas caracterizadas pelos postulados de Giuseppe Peano, já abordado nesse trabalho.

- **Funções Iniciais**: Uma função f: $W^n \rightarrow W^m$ é inicial se e somente se f for uma das funções a seguir:
  - Identidade i: $W^1 \rightarrow W^1$, definida por i(x) = x;
  - Zero z : $W^0 \rightarrow W^1$, definida por z( )=0;
  - Projeção π: $W^1 \rightarrow W^0$, definida por π(x)= <>;
  - Sucessor s: $W^1 \rightarrow W^1$, definida por s(x)= x + 1;

As funções podem ser combinadas em vários aspectos diferentes, um conjunto de funções é fechado sob algumas operações se a operação aplicada aos membros do conjunto sempre produzirem outros membros desse conjunto. A seguir serão mostradas as operações de Composição, Combinação, Expoentização e Repetição extraídas de[2][4]:

- **Composição**: Sejam as funções f: $W^r \rightarrow W^s$ e g: $W^s \rightarrow W^t$. A composição de f e g é a função g o f : W^r à W^t definida por: g o f(x_r) = g(f(x))
- **Combinação**: Sejam as funções f: $W^r \rightarrow W^s$ e g: $W^t \rightarrow W^u$. A combinação de f e g é a função g x f : $W^{r+t} \rightarrow W^{s+u}$ definida por: f x g(x_r , x_t)= < f(x_r) , g(x_t) >
- **Expoentização**: Sejam as funções f: $W^r \rightarrow W^r$. A expoentização de f é uma função f $^{\#}$: $W^{r+1} \rightarrow W^r$ definida por:

$$f^{\#}(x,y)= \overbrace{fo...of(x)}^{\text{Y vezes}}$$

As funções recursivas parciais contêm a classe de Funções Recursivas Primitivas acrescidas da operação de repetição, a qual é definida da seguinte forma:

- **Repetição**: Seja a função f: $W^{r+1} \rightarrow W^{r+1}$ a repetição de f (também chamada de minimização) , f ∇ : $W^{r+1} \rightarrow W^r$, é uma função definida por:

$$f\nabla(x_r, y) = \begin{cases} x'_r, \text{ sss para algum } k \geq 0 \\ f^{\#}(x_r,y,k) = < x_r',1 > \text{ e} \\ \text{para todo } l < k \text{ ,} \\ f^{\#}(x_r, y, k) \neq < x'_r, 1 >, \\ \text{indefinido se não existe tal} \\ k \geq 0 \end{cases}$$

A classe de Funções Recursivas Parciais é a menor classe contendo as funções iniciais e fechada sobre as operações de composição, combinação, expoentização e repetição. No contexto deste trabalho, por se tratar de um ambiente de aprendizado, estas funções não serão tratadas na sua implementação, que está restrita às funções primitivas recursivas, isto é, não será abordada a repetição.

A Linguagem Algorítmica Minimal (LAM)

A Linguagem Algorítmica Minimal (LAM) é baseada em [4] e dispõe de três tipos de comando de atribuição X←0, X←Y e X←X +1. A seguir será apresentado a forma Backus-Naur (BNF) desta linguagem.

<atribuição>::= <nome>←0|<nome>←<nome>|
<nome>←<nome> + 1
<nome>::=<literal>|<literal><numeral>
<literal>::=<letra>|<letra><literal>
<numeral>::=<algarismo>|<algarismo><numeral>
<algarismo>::=0|1|...|9
<programa io>::=<io><programa>
<entrada>::=ENTRADA|ENTRADA<lista de nomes>
<saida>::=SAIDA|SAIDA<lista de nomes>
<lista de nomes>::=<nome>|<nome>;<lista de nomes>
<programa>::=<comando rotulado>|<loop>|
<programa><end> | <programa><programa>
<comando rotulado>::=<comando>;|<comando>
<rotulo>;
<comando>::=<atribuição>|GOTO<rotulo>
<loop>::=LOOP<nome>;| <rotulo>LOOP<nome>;
<end>::=END;| <rotulo>END;

Um <programa io> é o que informalmente chamamos de uma programa LAM. Iniciando sempre com uma declaração das vaiáveis de entrada e saída, seguida por um corpo do programa.

Mapeando as Funções Recursivas Primitivas e Programas na Linguagem Algorítmica Minimal (LAM)

Para fins de simulação, neste artigo será abordada apenas a relação entre Funções Primitivas Recursivas e um subconjunto de LAM, que não contempla o comando GOTO. Desta forma, o aprendiz não terá dificuldades em entender estes dois modelos na sua versão mais simples e se prenderá a entender a relação entre os dois com a ajuda da ferramenta aqui proposta. A estratégia é exibir programas nessa linguagem Minimal que computem as funções primitivas recursivas definidas pelo aprendiz. Para mapear os operadores funcionais em programas em LAM, suponha que P_f seja o programa em LAM que compute uma função f: W→W e P_g o programa em LAM que compute uma função g: W→W. Este mapeamento é definido recursivamente e ilustrado na tabela a seguir.

Tabela 1: Mapeamento de funções iniciais

FUNÇÕES	PROGRAMAS EM LAM
i	ENTRADA X_1; SAIDA Y_1; $Y1 \leftarrow X_1$;
π	ENTRADA X_1; SAIDA ; $Y1_1 \leftarrow 0$;
z	ENTRADA; SAIDA Y_1; $Y_1 \leftarrow 0$;
s	ENTRADA X_1; SAIDA Y_1; $Y_1 \leftarrow X_1$; $Y_1 \leftarrow Y_1+1$;

Seja o programa em LAM que executa a função f: $W^n \rightarrow W^m$ tal que f(x,y) = (sos) x (ios).
ENTRADA A1, D2
SAÍDA C1, E2
B1 ← A1
B1 ← B1 + 1 (a)
C1 ← B1
C1 ← C1 +1 (b)
E2 ← D2
F2 ← E2
F2 ← F2 + 1 (c)
Neste exemplo, percebe-se que algumas das variáveis dependem de outras ao serem inicializadas. Utilizou-se o conceito de grafo de dependências para auxiliar o mapeamento e no entendimento de tais dependências.

Tabela 2: Mapeamento dos Operadores Funcionais

OPERADORES	PROG. EM LAM
Composição $f \circ g(x_1)$, onde $g(x_1) = y_g$ e $f(x_f) = y_1$	ENTRADA X1; SAIDA Y1; P_g $X_f \leftarrow Y_g$; P_f
Combinação $fxg(x_1,x_2)=$ $(f(x_1), g(x_2))$ onde $f(x_1) = y_1$ e $g(x_2) = y_2$	ENTRADA X_1,X_2; SAIDA Y_1,Y_2; P_f; P_g ou ENTRADA X_1,X_2; SAIDA Y_1,Y_2; P_g; P_f;
Exponentização $f^{\#}(x,y)$, onde $f(x) = z$	ENTRADA X,Y; SAIDA Y1; LOOP X P_f $X \leftarrow Z$; END

Grafo de Dependências

Um grafo de dependências [7] pode ser definido como um conjunto G = (V , A) direcionado, finito e não vazio, onde V representa o conjunto de vértices, e A, o conjunto de arestas que interligam estes vértices. É representado por unidades atômicas de execução de um programa, unidas por dependências de fluxo de dados e / ou controle, de modo a poder se representarem todas as dependências dentre as estruturas de processamento do programa (algoritmo). Nas instruções marcadas por (a), (b) e (c) acima, temos uma dependência de fluxo de modo que, depois de ser definida, tem seu valor utilizado posteriormente. O Grafo de dependências deste programa é representado na figura a seguir.

O grafo de dependência [7] é utilizado como uma linguagem intermediária, favorecendo o entendimento das dependências de fluxo de dados e de controle. Outra razão para se utilizar tal mecanismo foi o fato de este propiciar uma estrutura de dados compatível com as necessidades de implementação e ainda potencializar para trabalhos futuros a possibilidade de visualização no ambiente à relação entre os modelos por um grafo.

Trabalhos Correlatos

Na Computação, várias iniciativas na construção de ferramentas para auxiliar o ensino de disciplinas têm sido apresentadas. Dentre elas encontramos interpretadores que são utilizados nas disciplinas de Algoritmo e Linguagem de Programação. A utilização desse tipo de recurso de visualização permite a disposição de problemas mais complexos, atuando fortemente na motivação do aluno [10]. Um ambiente nesses moldes deve ter um analisador sintático

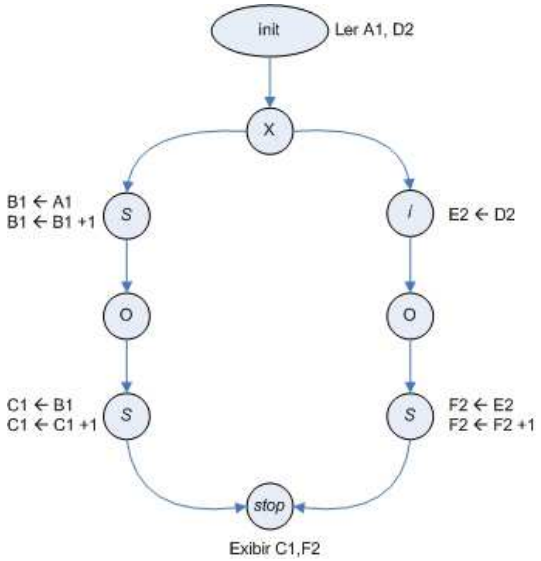

Figura 1: Função Ilustração do Grafo de Dependência associado ao exemplo 1.

integrado com um editor de texto. É desejável que eventuais erros sintáticos sejam identificados no processo de tradução, enquanto que as correções dos erros semânticos fiquem a encargo do aprendiz. [9] relata sobre a importância de visualização dos resultados na execução do algoritmo, pois, se tal operação fosse realizada em linguagem natural, a quantidade de variações de soluções em formatos diferentes seria muito grande. "A utilização de um instrumento computacional para a prática do mesmo, ao invés do desenvolvimento de algoritmos no papel, torna-se um referencial importante para o processo de ensino aprendizagem" [Nobre and Menezes 2002]. Não de agora, têm-se despendido esforços na construção de ferramentas que auxiliam os alunos em sua caminhada. Em [3] relata que, no Brasil, uma técnica de algoritmização baseada no conceito de PDL(Program Design Language) e chamada de Portugol, ou português estruturado, é mais utilizada.O Portugol é muito semelhante à Linguagem Pascal, só que em forma de pseudo-código.A diferença entre uma linguagem formal de programação e uma PDL é o fato de a última não ser normalmente executada. É o caso do ILA [6], que é um interpretador de linguagem algorítmica. Agregando-se ao ILA uma interface amigável com recursos gráficos e um simulador de memória, temos o AMBAP [1].Ainda no contexto de Algoritmo e Programação, tem-se o VisualG , que se assemelha muito ao Pascal, sendo que sua sintaxe está em PORTUGOL. Além dela, dentre outras ferramentas que se enquadram nesse contexto, podem-se citar o Portugol plus , o UAL e o Portugol IDE, porém este último ainda agrega, dentro dessa ferramenta, a possibilidade de visualização em um fluxograma.

Não obstante a essa diversidade, não se encontram com tanta facilidade simuladores ou interpretadores para a computabilidade das funções recursivas. [8] traz um compilador e simulador de funções recursivas parciais utilizando redes neurais. A diferença dessa ferramenta para a proposta neste artigo se dá pelo fato de que a do artigo citado lança mão de recursos de inteligência artificial, especificamente redes neurais para a sua implementação. Outra diferença deve-se à não geração de um código em linguagem algorítmica por parte da ferramenta citada, enquanto que a deste artigo assim o faz.

Arquitetura do Sistema

A figura 2 apresenta a tela inicial da ferramenta aqui proposta onde o usuário pode digitar a expressão ou escolher compor a expressão, utilizando os botões que estão explicitados na ferramenta. Nesses botões estão as opções das funções recursivas primitivas, dentre as quais estão a identidade, a projeção, o sucessor e o zero, além dos operadores de composição, de combinação e de expoentização. Uma vez que a execução for solicitada, o sistema gerará o pseudo-código.

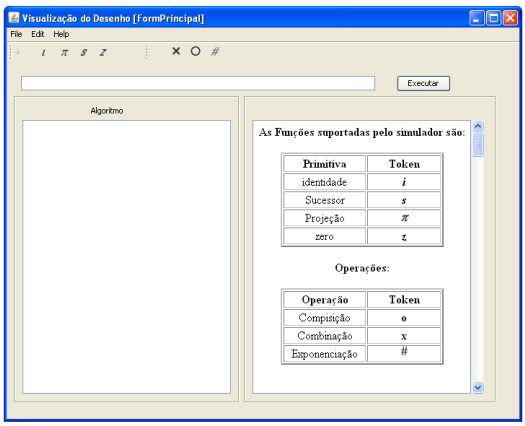

Figura 2: tela principal da ferramenta.

Quando o usuário entra com a expressão, esta será convertida para a notação posfixa por se tratar de uma notação com maior poder computacional em detrimento à infixa, que é a que os usuários estão mais habituados a trabalhar. É feita uma varredura na expressão para reconhecimento de padrões de expressões, tentando assim atenuar o trabalho de geração de código. Assim os padrões são comparados com expressões regulares para se encontrarem funções mais usuais, as quais foram pré-definidas, como a soma, a multiplicação, a diagonização entre outras, que estão armazenadas numa tabela de símbolos. Por fim, o algoritmo em LAM é exibido na tela principal. Todo este processo está descrito no fluxograma da figura 3.

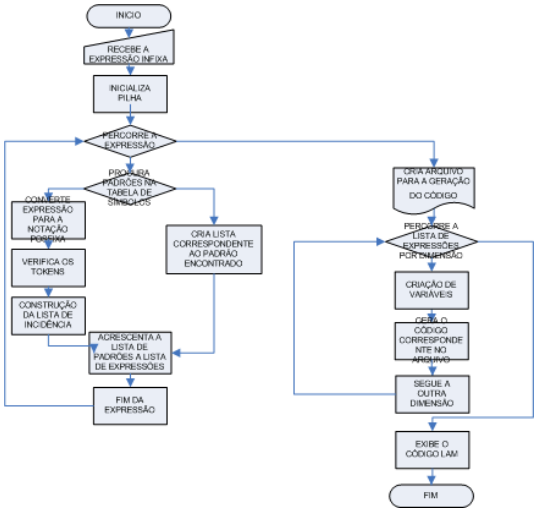

Figura 3: Fluxograma da ferramenta.

Ainda durante a varredura, o algoritmo se vale de listas de incidência dinâmicas para simular a construção de um grafo. A lista é construída aos moldes do grafo de dependência. Se for encontrado algum padrão na expressão, será gerada uma lista com o padrão correspondente, e logo em seguida tal lista será agregada à lista de incidência. Por fim, para a geração do código, é utilizado um algoritmo de Busca em profundidade na lista de incidência, pois a mesma se assemelha muito a uma árvore. Ao passo em que a estrutura é atravessada, o código é gerado em um arquivo que posteriormente será exibido.

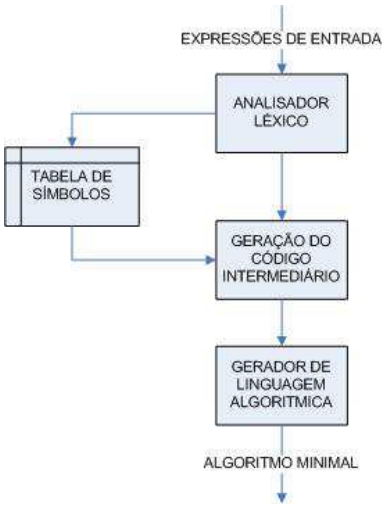

Figura 4: Arquitetura da ferramenta.

Como sugere a figura 4, este processo se assemelha muito ao processo de compilação, onde é realizada inicialmente uma análise lexical, com consultas a tabela de símbolos. O código intermediário é então gerado, o que neste artigo é representado pelo grafo de dependência e logo depois e gerado o código alvo.

Considerações Finais

Este artigo descreveu uma ferramenta para auxílio ao ensino de computabilidade, que se baseia na simulação entre os modelos das funções recursivas e uma linguagem de programação. A ferramenta foi concebida utilizando a linguagem JAVA. Tal escolha foi feita para melhor facilitar a integração e o acoplamento com o Ambiente ACARAjE citado anteriormente, tendo em vista que os outros módulos estão sendo desenvolvidos na mesma linguagem. A próxima etapa será de integração desta ferramenta com o Ambiente. Com o sistema proposto espera-se que os alunos de Teoria da Computação possam usufruir de um recurso computacional de fácil utilização que os auxilie na aprendizagem da disciplina, porém a consolidação da ferramenta só se dará depois de um feedback mais efetivo por parte dos usuários. Outra etapa considerada entre os trabalhos futuros seria a agregação junto à tela principal do sistema. Assim, permitiria uma visualização do grafo de dependência, de modo a facilitar a percepção da relação existente entre a função, o grafo e a linguagem algorítmica. Além disso, seria incorporada a implementação da função repetição dentre as parciais e, por conseguinte, o comando GOTO.

Referências

[1] E. S. Almeida and E. B. Costa and and Klebson S. Silva and R. B. Paes and A. A. M. Almeida and J. D. H. Braga, AMBAP: Um Ambiente de Apoio ao Aprendizado de Programação , *XXII Congresso da Sociedade Brasileira de Computação X WEI, Florianópolis SC*, (2002).

[2] W. S. Brainerd and L. H. Landweber, "Theory of Computation ", John Wiley & Sons, New York, 1974.

[3] P. L. L. Cares and R. L. S. Dazzi, "Ambiente para Teste de Mesa utilizando Fluxograma",em (Seminário de Computação e Feira de Informática SENAC),2002.

[4] R. L. Carvalho and C. M. G. M. Oliveira, "Modelos de Computação e Sistemas Formais ", Imprinta Gráfica e Editora Ltda, 1998.

[5] W. Carnieli and R. L. Epstein, "Computabilidade, Funções computáveis , Lógica e os Fundamenteos da Matemática ", Editora Unesp, 2006.

[6] J. Evaristo and S. Crespo, "Aprendendo a programar - programando numa linguagem algorítma executável ILA ", Book Express, 2000.

[7] J. Ferrante and K. J. Ottensteinand and J. D. Warren, The Program Dependence Graph and Its use in Optimization , *ACM Transactions on Programming Languages and Systems*, (1987) 319-349.

[8] J. Neto and J. F. Costa and P. Correia and M. Rosa , A compiler and simulator for partial recursive functions over neural networks , *Ahmad Lotfi and Jonathan M. Garibaldi (eds), Applications and Science in Soft Computing, Springer-Verlag*, (2004) 39-46.

[9] J. C. R. Pereira, "AVEP - Um Ambiente Virtual para apoio ao Ensino de Algoritmos e Programação Algoritmos e Programação ", Universidade Estadual Do Norte Fluminense , 2006.

[10] S. Pollard and J. Forbes, "UHands-on labs without computers",em (SIGCSE - 34th Technical Symposium on Computer Science Education),2003.

[11] B. L. Vieira and E. S. Almeida, "Um simulador para ilustrar graficamente a execução de programas e as noções de computabilidade", em (59ª Reunião da Sociedade Brasileira para o Progresso da Ciência),2007.