# PROGRAMAÇÃO LÓGICA NA MODELAGEM DE SÓLIDOS

Edgard Afonso Lamounier Júnior
Universidade Federal de Uberlândia
Departamento de Eng. Elétrica
Av. Universitária, s/n - Santa Mônica
38400 Uberlândia - MG
Tel : (034) 235-2888 - Ramal 180

Lília de Assunção Hess
Instituto Militar de Engenharia
Departamento de Informática
Pça. General Tibúrcio, 89 - Urca
22290 Rio de Janeiro - RJ
Tel : (021) 295-3232 - Ramal 214

Sérgio de Mello Schnider
Universidade Federal de Uberlândia
Coord. Pós-Graduação em Eng. Elétrica
Av. Universitária, s/n - Santa Mônica
38400 Uberlândia - MG
Tel (034) 235-2888 - Ramal 180

## RESUMO

Este trabalho trata da construção de um modelador de sólidos em PROLOG e analisa a maneira de descrever um esquema de representação CSG em linguagem de programação lógica.

## 1 - INTRODUÇÃO

Atualmente, nota-se que sistemas computacionais que tratam da geometria de sólidos rígidos, tornam-se cada vez mais importantes em áreas como robótica, alguns ramos da engenharia, sistemas de

CAD/CAM e outras áreas que lidam com o fenômeno espacial.

Para modelar sólidos rígidos, adotamos entidades geométricas abstratas, que são elementos do Espaço Euclideano ($\mathbb{R}^3$), chamadas de *r-sets*, as quais são consideradas modelos adequados para este tipo de representação.

Dado um certo *r-set*, com seu significado geométrico(semântica), existe uma função, que o liga a uma estrutura construída com símbolos de um alfabeto de acordo com regras sintáticas, chamada de esquema de representação [1].

Dos esquemas de representação conhecidos, o adotado para este trabalho foi Geometria Construtiva de Sólidos("Constructive Solid Geometry, CSG"). Basicamente, representações CSG são árvores binárias cujos nódulos não-folhas são operadores(união, interseção, diferença) e cujos nódulos folhas são sólidos primitivos ou sólidos gerados pela combinação de dois outros. A raiz da árvore é o sólido modelado.

Notamos que a maioria dos modeladores de sólidos baseados em CSG [2], [3], bem como em outros esquemas de representação, são desenvolvidos em linguagens de programação procedurais, onde os processos que tratam as representações dos sólidos são feitos através de uma sequência precisa de códigos e instruções, e geram uma diferença significativa entre a maneira como as pessoas raciocinam sobre sólidos e a maneira em que os mesmos são descritos.

Por outro lado, diversas pesquisas têm sido feitas na tentativa de se desenvolver projetos de geometria computacional utilizando-se linguagens não-procedurais, pois elas provêem grande interação e mecanismos poderosos de processamento de dados. Em particular, a linguagem de programação lógica PROLOG é uma boa ferramenta para implementar algoritmos computacionais geométricos [4], [5], [6]. Por exemplo, foi verificado numa implementação de interseção de polígonos que PROLOG é muito mais compacto e mais fácil de usar do que FORTRAN. Também, por sua natureza, esta linguagem de programação facilita a construção de árvores CSG na modelagem de sólidos, como veremos a seguir.

## 2 - PROGRAMAÇÃO LÓGICA E DESCRIÇÃO DE SÓLIDOS

A linguagem de programção PROLOG é largamente aceita para propósitos da computação simbólica. Um programa PROLOG é a descrição de um problema na forma de algumas declarações e regras sobre como a solução pode ser deduzida de fatos; isto é feito baseando-se nas cláusulas de Horn do Cálculo de Predicados. O programador somente descreve que problema deve ser resolvido e não se preocupa em como a solução é executada pela máquina. A unificação de fórmulas lógicas é um operador primitivo e a lista é a única estrutura de dados.

Como um simples exemplo do uso de PROLOG em geometria, vamos considerar o problema de ligar arestas isoladas e sua solução nesta linguagem :

```
                x  y  z
vertice(v1,1,2,3).          aresta(a1,v1,v2).
vertice(v2,4,6,7).          aresta(a2,v2,v3).
vertice(v3,6,4,8).          aresta(a3,v3,v4).
vertice(v4,5,6,3).

                                  lista de arestas
liga_arestas([]).
liga_arestas([C|R]) :-
     aresta(C,V1,V2),   → encontra os vértices que compõe a aresta
     liga_vertices(V1,V2),
     liga_arestas(R).

liga_vertices(V1,V2) :-
     vertice(V1,X1,Y1,Z1),
     vertice(V2,X2,Y2,Z2),
     line(X1,Y1,Z1,X2,Y2,Z2).
```

As cláusulas vertice e aresta são fatos indicando respectivamente o nome de um vértice com suas coordenadas em 3D e o nome de uma aresta seguida dos vértices que a formam. A cláusula liga-arestas tem como argumento a lista de arestas, escrita da forma [C|R], separando a primeira aresta do restante. Quando o

programa atinge a cláusula aresta(C,V1,V2) é feita uma unificação com a base de fatos obtendo-se assim os vértices V1 e V2 que formam a aresta C. A cláusula liga-vertices(V1,V2) procede de maneira semelhante para obter as coordenadas dos vértices V1 e V2 ligando-os com a cláusula line, que traça uma linha entre os pontos (X1,Y1,Z1) e (X2,Y2,Z2) do espaço. Quando a lista de arestas for vazia o programa atinge a cláusula liga-aresta([]) e termina o processamento.

Um outro bom exemplo é o problema de certificar se uma dada aresta esta inclusa(pertence) ao conjunto das arestas conectadas, que pode ser resolvido assim:

```
pertence(A,[A|_]).
pertence(A,[_|R]) :- pertence(A,R).
```

Uma abordagem mais completa sobre programação lógica e PROLOG, podem ser encontrados em [7] e [8].

## 2.1 - Descrição Lógica dos r-sets

As implicações matemáticas dos *r-sets* são discutidas em [1], onde é argumentado que eles são apropriados para modelar sólidos físicos pelo fato de serem conjuntos do $\mathbb{R}^3$ que são limitados, fechados, regulares e semianalíticos. De uma forma mais simples podemos dizer que um *r-set* é formado pela superfície mais o conjunto de pontos interiores de um volume e não pode ter arestas "penduradas".

Para utilizar os *r-sets* na implementação deste trabalho, nós o construímos através da união de *r-sets* bidimensionais(faces), mais a informação de seu interior. Portanto, para descrever os sólidos primitivos usados para gerar outros sólidos através de uma CSG nos consideramos três itens, a saber :

(a) o sólido primitivo propriamente dito.

(b) cada face que constiui o sólido.

(c) cada vértice de uma face.

A estrutura de dados do sólido consiste de seu nome, uma lista de seus vértices e uma lista de suas faces.

A estrutura de dados da face contém o nome da mesma e uma lista de seus vértices ordenados no sentido horário. Esta orientação é usada para determinar o interior do sólido através do produto vetorial das arestas([9]).

A estrutura de dados do vértice , contém seu nome, sua localização espacial e uma lista de vértices adjacentes.

Tomando como instância deste sólido um cubo de lado 3, o mesmo fica assim descrito em predicados PROLOG :

```
/* Descricao do Solido */
solido(cubo,[v1,v2,v3,v4,v5,v6,v7,v8],[f1,f2,f3,f4,f5,f6]).
/* Descricao das Faces */
face(f1,[v1,v2,v3,v4]).
face(f2,[v1,v8,v7,v2]).
face(f3,[v2,v7,v6,v3]).
face(f4,[v7,v8,v5,v6]).
face(f5,[v3,v6,v5,v4]).
face(f6,[v1,v8,v5,v4]).

/* Descricao dos Vertices */
vertice(v1,[5.0,8.0,8.0],[v2,v4,v8]).
vertice(v2,[8.0,8.0,8.0],[v3,v7]).
vertice(v3,[8.0,5.0,8.0],[v4,v6]).
vertice(v4,[5.0,5.0,8.0],[v5]).
vertice(v5,[5.0,5.0,5.0],[v6,v8]).
vertice(v6,[8.0,5.0,5.0],[v7]).
vertice(v7,[8.0,8.0,5.0],[v8]).
vertice(v8,[5.0,8.0,5.0],[]).
```

Note que na descrição dos vertices não há repetição da conexão de vértices, evitando-se assim uma redundância na formação das arestas.

## 2.2 - Construção Lógica de Uma CSG

Nesta seção temos um exemplo de uma árvore CSG (Figura 1) e logo abaixo, a maneira como a mesma pode ser descrita em predicados PROLOG.

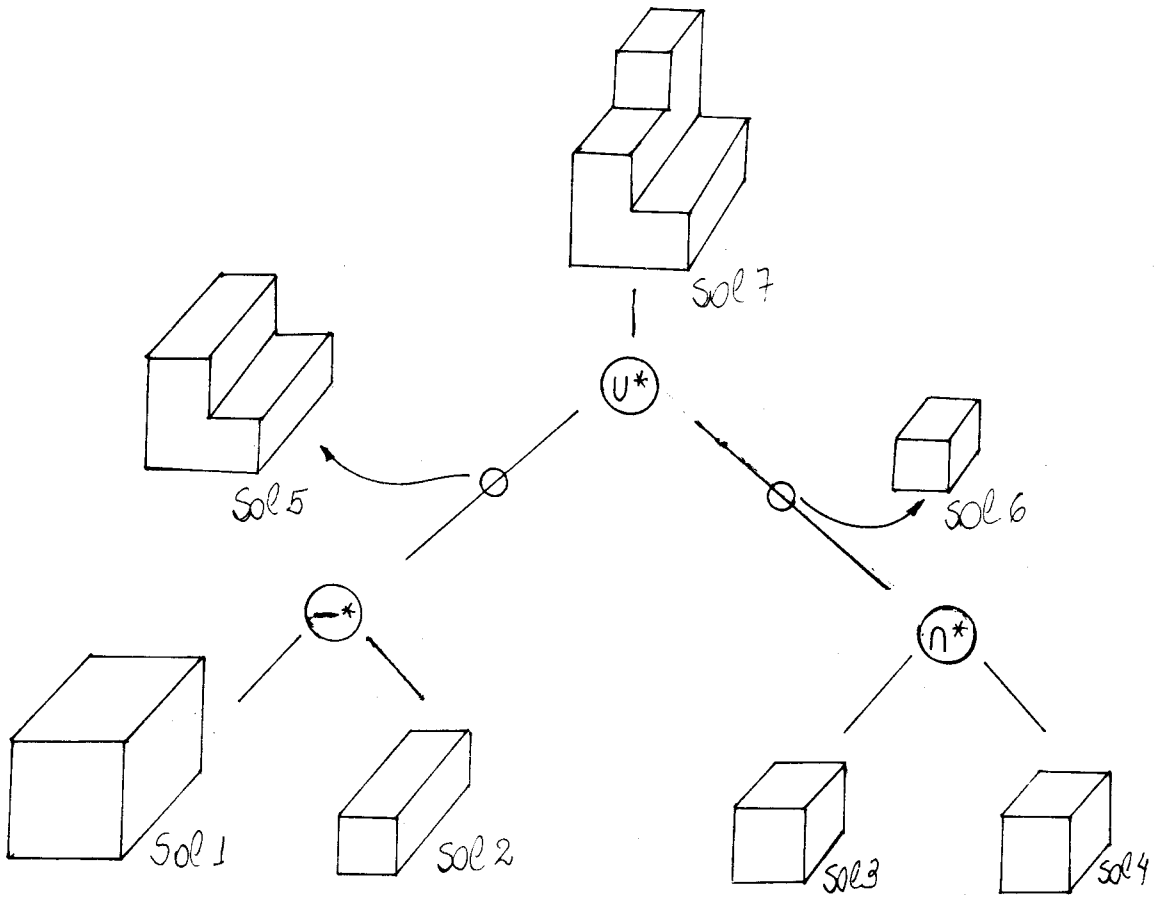


Fig. 1 - Uma árvore CSG.

```
modela_sólido(Sol7) :-
        diferença_regularizada(Sol1,Sol2,Sol5),
        intersecao_regularizada(Sol3,Sol4,Sol6),
        uniao_regularizada(Sol5,Sol6,Sol7).
```

Note como o PROLOG facilita a construção de uma CSG, e que podemos ter diferentes tipos de sólidos gerados, bastando apenas

alterar a base de fatos.

## 3 - UM MODELO AUXILIAR PARA IMPLEMENTAÇÃO

É essencial que os conjuntos dos sólidos estejam fechados sobre as operações que são executadas sobre eles para garantir que os resultados possam ser usados posteriormente em outras operações. Como os operadores padrões de conjuntos não preservam a solidez é necessário substituí-los por suas versões regularizadas(os *r-sets* são fechados apenas sobre as operações regularizadas) A teoria sobre o conjunto de operadores regularizados( $\cup^*$, $\cap^*$, $-^*$ ) é bem entendida e pode ser encontrada em [1], [10] e [11].

### 3.1 - Classificação de Pertinência

Para executar uma operação regularizada sobre dois sólidos, nosso algorítmo faz uma classificação de pertinência proposta por Robert B. Tilove.

Os *r-sets* serão determinados através do resultado da classificação entre dois conjuntos de pontos, o conjunto candidato e o conjunto referência. A classificação M(X,S) de um conjunto candidato X em relação a um conjunto referência S é a segmentação de X em subconjuntos XinS, XonS e XoutS, que são conjuntos que estão no interior, na fronteira e no exterior de S respectivamente. A Figura 2, mostra a classificação de um conjunto X em relação a um conjunto S.

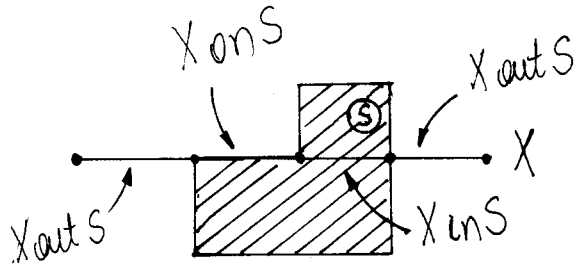


Fig. 2 - Classificação de Pertinência

O problema consiste em : dados dois objetos A e B e sua classificação em relação a X, ou seja, (XinA, XonA, XoutA) e (XinB, XonB, XoutB), encontrar (XinS, XonS, XoutS), onde S = A ⊗ B e ⊗ denota um dos operadores regularizados. Na Figura 3, vemos um exemplo simples. Mais informações sobre classificação de membros podem ser encontradas em [10], [11] e [12].

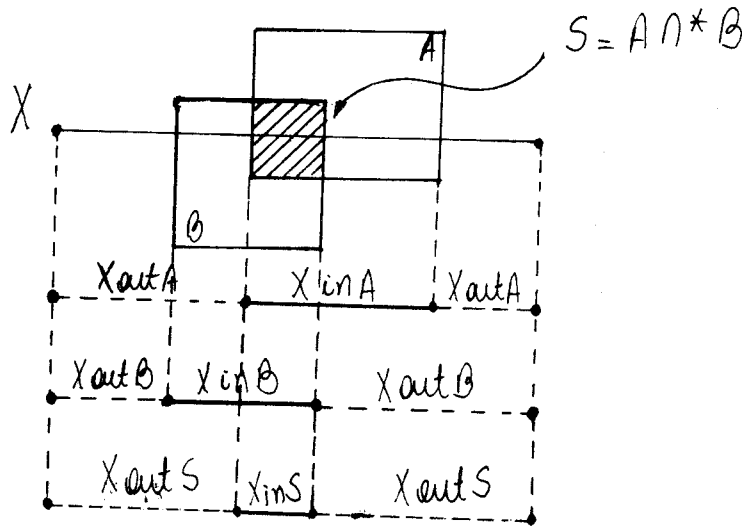


Fig. 3 - Combinando classificações.

Para gerar um sólido S = A ⊗ B, nosso algoritmo combinando suas classificações em relação a uma linha candidata X segundo a operação ⊗, completa a informação do interior de S através de análise da classificação nos três planos de projeções, reconstruindo este sólido em 3D. No Apêndice, vemos um exemplo mostrando a interseção de dois sólidos exibidos através de representação "wireframe" (porque correntemente, o tratamento de *r-set* não foi aproveitado completamente para a representação de saída dos resultados no display).

## 4 - CONCLUSÃO

Foi apresentado um modelo de representação lógica de sólidos, construindo-se árvores CSG através da linguagem de programação PROLOG. Dois sólidos são operacionalizados através da classificação de membros em relação a um conjunto candidato e da

combinação destas classificações.

Para modelar sólidos físicos, são adotados *r-sets*, definidos através da união de *r-sets* bidimensionais mais a informação de pontos interiores.

O trabalho é implementado com o interpretador Arity PROLOG, versão 4.0, em um microcomputador compatível com o IBM-PCxt. O sistema gráfico adotado para este programa de aplicação é o GKS ([6]).

## REFERÊNCIAS


[1]. Requicha, A. A. G., and Voelker, Hebert B.; *"Representations for Rigid Solids : Theory, Methods and Systems"*, ACM Computing Surveys, Vol. 12, No. 4 p. 437-464, Dezembro, 1980.

[2]. Laidlaw, David H. and Trumbore, W. Benjamim; *"Constructive Solid Geometry for Polyedral Objects"*, Computer Graphics ACM/SIGGRAPH, Vol. 20, NO. 4, p.161-170, Agosto, 1986.

[3]. Longhi, Magali T., Sasso-Freitas, Carla D. M. e Golendziner, Lia G.; *"N-MOS : Um Núcleo de Modelagem de Sólidos Baseado em Octrees"*, Anais do XX Congresoo Nacional de Informática, SUCESU, 1987.

[4]. Franklin, Wm. Randolph; *"Computational Geometry and PROLOG"*, Fundamentals Algorithms for Computer Graphics, Springer-Verlag, Berlin, 1985.

[5]. Franklin, Wm. Randolph, Wu, Peter Y. F., Samadar, Samiro and Nichols, Margaret; *"PROLOG and Geometry Projects"*, IEEE CG&A, p. 46-55, Novembro, 1986.

[6]. Oliveira Neto, Edgard C., *"Uma Proposta para Programação Lógica Gráfica"*, Tese de Mestrado, IME, 1988.

[7]. Davis, R. E., *"Logic Programming and PROLOG : A Tutorial"*, IEEE Software Vol. 2, No. 5, p. 53-62, Setembro, 1985.

[8]. Bratko, L.; *"PROLOG Programming for Artificial Intelligence"*, Addison-Wesley, Inglaterra, 1986.

[9]. Harrington, S.; *"Computer Graphics, A Programming Approach"*, McGraw-Hill, Nova Iorque, 1983.

[10]. Tilove, R. B., *"Set Membership Classification : A Unified Approach to Geometric Intersection Problems"*, IEEE Trans. Comput., Vol. C-29, No. 10, p. 874-883, Outubro, 1980.

[11]. Requicha, A. A. G., and Voelker, Hebert B., *"Boolean Operations in Solid Modeling : Boundary Evaluation and Merging Algorithms"*, Proceedings of the IEEE, Vol. 73, No. 1, p. 30-44, Janeiro, 1985.

[12]. Lamounier, Edgard A. Jr., Hess, Lilia A., *"Um Modelador Lógico de Sólidos"*, Anais do 5º Simpósio Brasileiro de Inteligência Artificial, Natal, 1988.

APÊNDICE

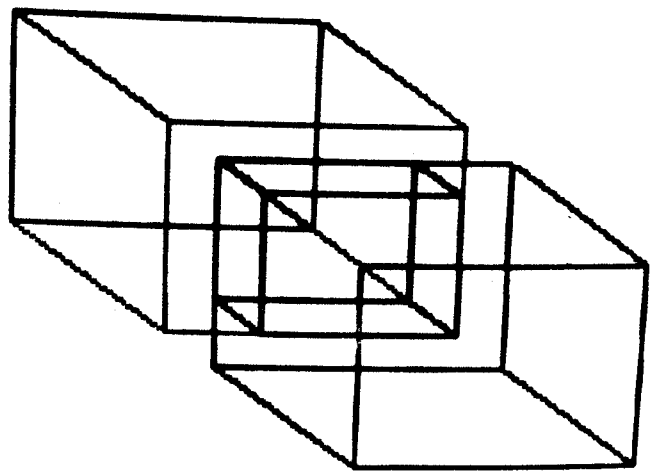

Fig. 4 - Interseção de dois sólidos em "wireframe".