Diagramas de Decisão Binária

Silvio do Lago Pereira e Leliane Nunes de Barros

Instituto de Matemática e Estatística – Universidade de São Paulo
{slago,leliane}@ime.usp.br,

Resumo Nesse artigo, usamos a expansão de Shannon para converter uma expressão booleana para uma forma canônica, denominada *forma normal condicional*, que descreve uma *árvore de decisão* para essa expressão. Em seguida, mostramos como um *diagama de decisão binária* pode ser obtido a partir da otimização de uma árvore de decisão, definimos algumas operações sobre essa estrutura de dados e apresentamos uma implementação em Prolog.

1 Expressões booleanas

Formalmente, a sintaxe de uma expressão booleana ε é definida pela gramática

$$\varepsilon ::= 0 \mid 1 \mid x_i \mid \neg\varepsilon \mid \varepsilon \wedge \varepsilon \mid \varepsilon \vee \varepsilon \mid \varepsilon \rightarrow \varepsilon \mid \varepsilon \leftrightarrow \varepsilon, \tag{1}$$

onde 0 e 1 denotam, respectivamente, as constantes *falso* e *verdade*; x_i denota uma variável proposicional; e os operadores $\neg$, $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$ denotam, respectivamente, *negação*, *conjunção*, *disjunção*, *implicação* e *bi-implicação* .

Por convenção, associamos prioridades aos operadores (em ordem decrescente: $\neg$, $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$) e, para resolver ambigüidades ou alterar a prioridade relativa entre eles, utilizamos parênteses.

Sejam ε uma expressão booleana e v uma *valoração* para ε (*i.e.* um mapeamento que associa a cada variável $x_i \in \varepsilon$ um valor $v(x_i) \in \{0, 1\}$). Então, o valor de ε pode ser definido, indutivamente, da seguinte maneira:

- $v(\neg\varepsilon) = 0 \Longleftrightarrow v(\varepsilon) = 1$
- $v(\varepsilon_1 \wedge \varepsilon_2) = 1 \Longleftrightarrow v(\varepsilon_1) = v(\varepsilon_2) = 1$
- $v(\varepsilon_1 \vee \varepsilon_2) = 0 \Longleftrightarrow v(\varepsilon_1) = v(\varepsilon_2) = 0$
- $v(\varepsilon_1 \rightarrow \varepsilon_2) = 0 \Longleftrightarrow v(\varepsilon_1) = 1$ e $v(\varepsilon_2) = 0$
- $v(\varepsilon_1 \leftrightarrow \varepsilon_2) = 1 \Longleftrightarrow v(\varepsilon_1) = v(\varepsilon_2)$

2 Expansão de Shannon

Sejam ε uma expressão booleana e $\varepsilon[c/x]$ a expressão obtida a partir de ε, substituindo-se toda ocorrência da variável x pela constante $c \in \{0, 1\}$. A *expansão de Shannon* da expressão ε, com relação à variável x, é dada por

$$\varepsilon \equiv \varepsilon[1/x] \vee \varepsilon[0/x]. \tag{2}$$

Com base nessa equivalência, definimos o operador condicional *ite*[1] como

$$ite(x, \varepsilon, \varepsilon') = (x \wedge \varepsilon) \vee (\neg x \wedge \varepsilon'), \tag{3}$$

ou seja, $ite(x, \varepsilon, \varepsilon')$ é verdade se o teste x e a expressão ε são verdadeiros ou se o teste x é falso e a expressão ε' é verdadeira.

Todo operador utilizado na gramática (1) pode ser expresso por meio do operador condicional *ite*, veja:

- $\neg x = ite(x, 0, 1)$
- $x_1 \wedge x_2 = ite(x_1, 1 \wedge x_2, 0 \wedge x_2) = ite(x_1, x_2, 0) = ite(x_1, ite(x_2, 1, 0), 0)$
- $x_1 \vee x_2 = ite(x_1, 1 \vee x_2, 0 \vee x_2) = ite(x_1, 1, x_2) = ite(x_1, 1, ite(x_2, 1, 0))$
- $x_1 \rightarrow x_2 = ite(x_1, 1 \rightarrow x_2, 0 \rightarrow x_2) = ite(x_1, x_2, 1) = ite(x_1, ite(x_2, 1, 0), 1)$
- $x_1 \leftrightarrow x_2 = ite(x_1, 1 \leftrightarrow x_2, 0 \leftrightarrow x_2) = ite(x_1, ite(x_2, 1, 0), ite(x_2, 0, 1))$

3 Forma normal condicional

Uma expressão booleana está na *forma normal condicional* se e só se contém apenas constantes, variáveis e o operador condicional *ite*, com todos os testes realizados sobre variáveis e com variáveis ocorrendo apenas como testes [1].

Toda expressão booleana ε pode ser convertida, indutivamente, para a forma normal condicional:

- se ε só contém variáveis de teste, ela já está na forma normal condicional;
- senão, enquanto houver uma variável $x \in \varepsilon$ que não seja teste, reescreva ε como $ite(x, \varepsilon[1/x], \varepsilon[0/x])$.

Como exemplo, vamos converter a expressão booleana $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$ para a forma normal condicional:

$$\begin{aligned}
&(x_1 \vee x_2) \wedge (x_2 \vee x_3) \\
&= ite(x_1, (1 \vee x_2) \wedge (x_2 \vee x_3), (0 \vee x_2) \wedge (x_2 \vee x_3)) \\
&= ite(x_1, x_2 \vee x_3, x_2) \\
&= ite(x_1, ite(x_2, 1 \vee x_3, 0 \vee x_3), ite(x_2, 1, 0)) \\
&= ite(x_1, ite(x_2, 1, x_3), ite(x_2, 1, 0)) \\
&= ite(x_1, ite(x_2, 1, ite(x_3, 1, 0)), ite(x_2, 1, 0))
\end{aligned}$$

4 Árvore de decisão

A forma normal condicional de uma expressão booleana descreve um grafo, denominado *árvore de decisão*, que define o valor dessa expressão sob toda

[1] *if-then-else*

valoração possível de suas variáveis. Por exemplo, a árvore de decisão correspondente à expressão $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$, cuja forma normal condicional é $ite(x_1, ite(x_2, 1, ite(x_3, 1, 0), ite(x_2, 1, 0))$, pode ser vista na figura 1.

Numa árvore de decisão, folhas são rotuladas com constantes e os demais nós são rotulados com variáveis. Ademais, cada nó interno x_i tem um filho esquerdo (assumindo $x_i = 1$) e um filho direito (assumindo $x_i = 0$). Na representação gráfica da árvore de decisão, os filhos esquerdo e direito são indicados através de linhas contínuas e pontilhadas, respectivamente.

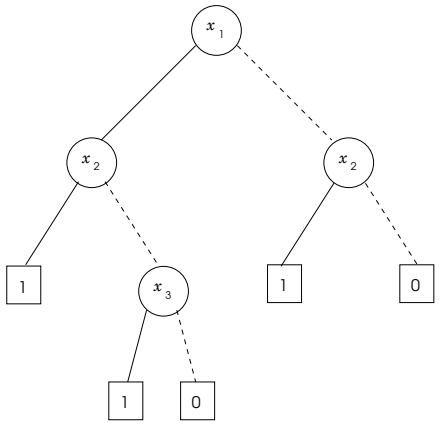

Figura 1. Árvore de decisão para a expressão $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$

Embora árvores de decisão sejam um dispositivo muito útil para representar funções booleanas, freqüentemente, elas podem apresentar muita redundância. Veja, por exemplo, a árvore de decisão para a expressão $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$ (figura 2), cuja conversão para a forma normal condicional é apresentada a seguir:

$$
\begin{aligned}
&(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4) \\
&= ite(x_1, (1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4), (0 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)) \\
&= ite(x_1, x_2 \wedge (x_3 \leftrightarrow x_4), \neg x_2 \wedge (x_3 \leftrightarrow x_4)) \\
&= ite(x_1, ite(x_2, 1 \wedge (x_3 \leftrightarrow x_4), 0 \wedge (x_3 \leftrightarrow x_4)), \neg x_2 \wedge (x_3 \leftrightarrow x_4)) \\
&= ite(x_1, ite(x_2, x_3 \leftrightarrow x_4, 0), ite(x_2, 0 \wedge (x_3 \leftrightarrow x_4), 1 \wedge (x_3 \leftrightarrow x_4))) \\
&= ite(x_1, ite(x_2, x_3 \leftrightarrow x_4, 0), ite(x_2, 0, x_3 \leftrightarrow x_4)) \\
&= ite(x_1, ite(x_2, ite(x_3, 1 \leftrightarrow x_4, 0 \leftrightarrow x_4), 0), ite(x_2, 0, ite(x_3, 1 \leftrightarrow x_4, 0 \leftrightarrow x_4))) \\
&= ite(x_1, ite(x_2, ite(x_3, x_4, \neg x_4), 0), ite(x_2, 0, ite(x_3, x_4, \neg x_4))) \\
&= ite(x_1, ite(x_2, ite(x_3, ite(x_4, 1, 0), \neg x_4), 0), ite(x_2, 0, ite(x_3, ite(x_4, 1, 0), \neg x_4))) \\
&= ite(x_1, ite(x_2, ite(x_3, ite(x_4, 1, 0), ite(x_4, 0, 1)), 0), \\
&\qquad ite(x_2, 0, ite(x_3, ite(x_4, 1, 0), ite(x_4, 0, 1))))
\end{aligned}
$$

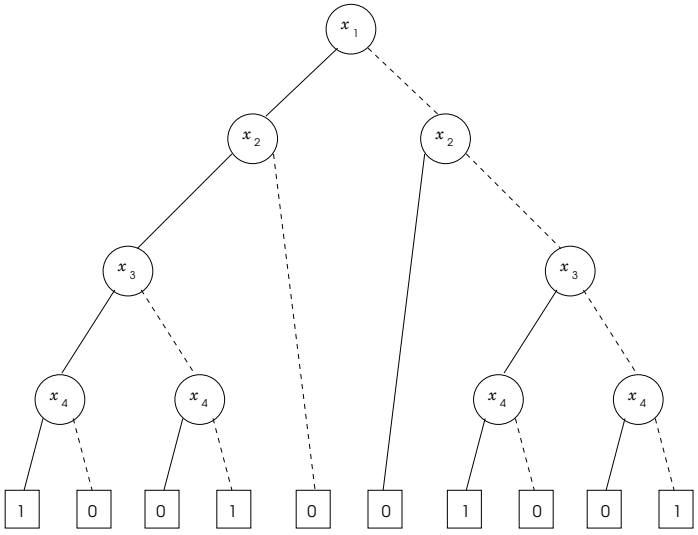

Figura 2. Árvore de decisão para a expressão $(x_1 \leftrightarrow x_2) \wedge (x_3 \leftrightarrow x_4)$

5 Diagrama de decisão binária

Como podemos observar na figura 2, parte da redundância existente numa árvore de decisão pode ser eliminada através do compartilhamento de subgrafos isomorfos (*e.g.* os subgrafos para as expressões $ite(x_4, 1, 0)$ e $ite(x_4, 0, 1)$ aparecem duplicados). De fato, quando todo subgrafo isomorfo é compartilhado, a árvore de decisão é transformada num grafo dirigido acíclico, denominado *diagrama de decisão binária* (BDD).

Particularmente, quando a ordem das variáveis de teste nos caminhos que levam da raiz até uma folha é sempre a mesma, o grafo obtido pelo compartilhamento de subgrafos isomorfos é denominado *diagrama de decisão binária ordenado* (OBDD). A figura 3 mostra o resultado do compartilhamento dos subgrafos isomorfos da árvore apresentada na figura 2.

Às vezes, após o compartilhamento de subgrafos, alguns testes podem se tornar redundantes. Para eliminar um teste redundante, basta excluir o nó que representa esse teste e redirecionar todo arco de entrada desse nó para o seu filho (figura 4). Quando todos os testes redundantes num diagrama de decisão binária ordenado são eliminados, o grafo resultante é denominado *diagrama de decisão binária ordenado reduzido* (ROBDD) [2].

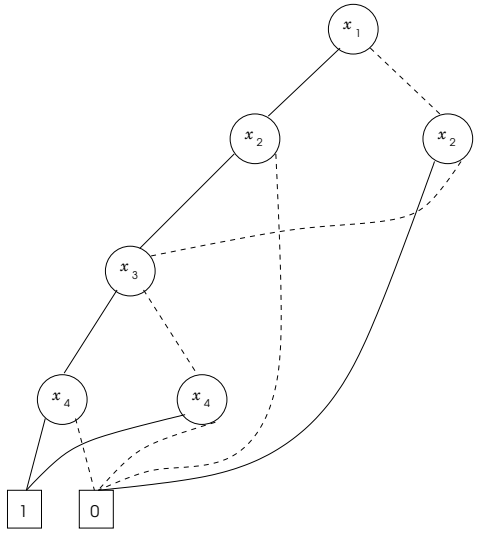

Figura 3. Compartilhamento de grafos isomorfos na árvore de decisão da figura 2

Os ROBDD's apresentam algumas propriedades importantes [1]:

- para toda expressão booleana há um único ROBDD correspondente;
- proporcionam uma representação compacta para expressões booleanas;
- possibilitam algoritmos muito eficientes para manipulação das expressões.

6 Algoritmos para manipulação de ROBDD's

Em termos de estruturas de dados, um ROBDD é uma tabela $T : n \mapsto \langle v, t, f \rangle$, que associa a cada identificador n um nó com variável de teste v, filho esquerdo t e filho direito f. Ademais, devido ao compartilhamento de subgrafos, a tabela T tem uma inversa $T^{-1} : \langle v, t, f \rangle \mapsto n$, mapeando nós em identificadores, que será utilizada para garantir que os diagramas sejam reduzidos. Também assumiremos que $T(n) = T^{-1}(\langle v, t, f \rangle) = nil$, sempre que $(n, \langle v, t, f \rangle) \notin T$.

Nessas tabelas, os identificadores são $0, 1, 2, \ldots$ (sendo 0 e 1 reservados para os nós terminais), as variáveis $x_1, x_2, \ldots$ são representadas pelos índices $1, 2, \ldots$ e a ordem em que as variáveis são testadas é definida pelos seus índices.

Para facilitar a representação dos algoritmos, trataremos a tabela T como uma variável global e escreveremos $|T|$ para denotar o número de entradas existentes na tabela T.

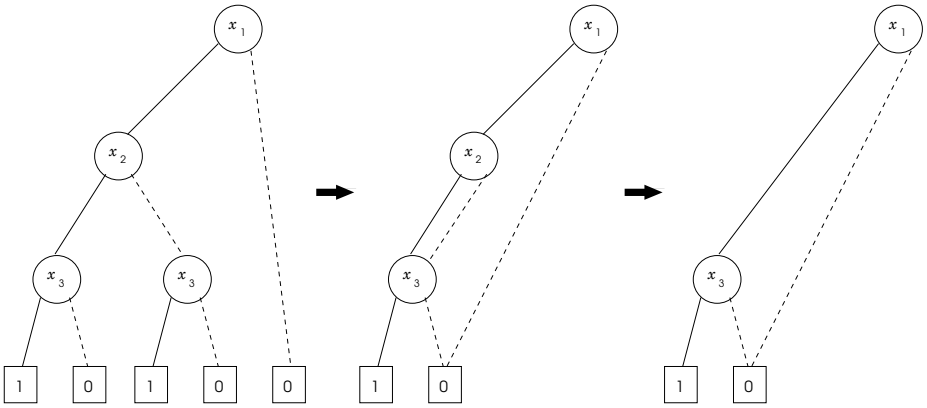

Figura 4. Compartilhamento de grafos isomorfos e eliminação de testes redundantes

6.1 Inicialização da estrutura

O algoritmo INIT recebe uma entrada m, indicando o número máximo de variáveis existentes na expressão booleana a ser representada, e inicia tabela T com duas tuplas especiais, representando os nós terminais 0 e 1. Ademais, para garantir a uniformidade no tratamento dos nós do ROBDD, os terminais são associados à variável x_{m+1}.

INIT$[T](m)$
1 $T \leftarrow \{(0, \langle m+1, nil, nil\rangle), (1, \langle m+1, nil, nil\rangle)\}$

6.2 Inserção de nós

Inicializada a estrutura, podemos inserir um nó usando o algoritmo INS:

INS$[T](v, t, f)$
1 **se** $t = f$ **então devolva** t
2 $n \leftarrow T^{-1}(\langle v, t, f\rangle)$
3 **se** $n = nil$ **então**
5 $n \leftarrow |T|$
6 $T \leftarrow T \cup \{(n, \langle v, t, f\rangle)\}$
7 **devolva** n

Quando tentamos inserir um nó, caso esse nó seja redundante, a função INS simplesmente devolve o identificador de seu filho; caso esse nó já tenha sido criado anteriormente, a função devolve seu identificador e, finalmente, caso o nó seja novo, a função o cria e devolve seu identificador.

6.3 Construção do diagrama de decisão

O algoritmo BUILD recebe uma expressão na forma normal condicional, cria uma tabela com os nós do diagrama de decisão reduzido para essa expressão e devolve o identificador para o nó raiz desse diagrama.

BUILD($ite(v, \varepsilon_t, \varepsilon_f)$)
1 **se** $\varepsilon_t, \varepsilon_f \in \{0, 1\}$ **então devolva** INS($v, \varepsilon_t, \varepsilon_f$)
2 **se** $\varepsilon_f \in \{0, 1\}$ **então devolva** INS(v,BUILD(ε_t), ε_f)
3 **se** $\varepsilon_t \in \{0, 1\}$ **então devolva** INS(v, ε_t,BUILD(ε_f))
4 **devolva** INS(v,BUILD(ε_t),BUILD(ε_f))

Na figura 5, podemos ver como a execução do algoritmo BUILD, para a expressão $ite(x_1, ite(x_2, ite(x_3, 0, 1), 1), ite(x_3, 1, 0))$, dispara chamadas ao algoritmo INS, que insere nós no diagrama sendo construído (observe que a construção é feita seguindo uma estratégia de busca *depth-first*).

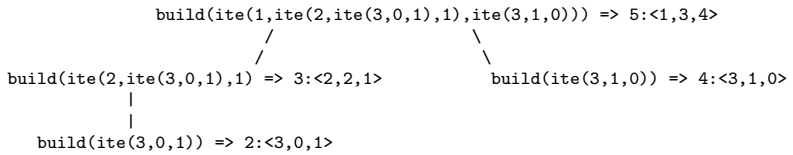

Figura 5. Inserções em T, realizadas pela execução do algoritmo BUILD

6.4 Operações booleanas entre diagramas

Todas as operações booleanas são implementadas pelo mesmo algoritmo genérico APPLY, que baseia-se na seguinte equivalência:

$$ite(x, \varepsilon_t, \varepsilon_f)\, op\, ite(x, \varepsilon'_t, \varepsilon'_f) \equiv ite(x, \varepsilon_t\, op\, \varepsilon'_t, \varepsilon_f\, op\, \varepsilon'_f) \quad (4)$$

Assim, para efetuarmos uma operação booleana entre duas expressões, basta aplicarmos essa transformação, recursivamente, a partir das raízes dos diagramas dessas expressões. Para maior eficiência, o algoritmo apresentado a seguir utiliza a técnica de *memoização*[2], implementada por meio da tabela M.

APPLY(op, r_1, r_2)
1 $M \leftarrow \emptyset$
2 **devolva** APPLY$'$(op, r_1, r_2)

[2] Programação dinâmica sob demanda.

$\textsc{Apply}'[T, M](op, r_1, r_2)$
1 se $M(\langle r_1, r_2\rangle) \neq nil$ então devolva $M(\langle r_1, r_2\rangle)$
2 $\langle v_1, t_1, f_1\rangle \leftarrow T(r_1)$
3 $\langle v_2, t_2, f_2\rangle \leftarrow T(r_2)$
4 se $r_1, r_2 \in \{0, 1\}$ então $r \leftarrow op(r_1, r_2)$
5 senão se $v_1 = v_2$ então $r \leftarrow \textsc{Ins}(v_1,\textsc{Apply}'(op, t_1, t_2),\textsc{Apply}'(op, f_1, f_2))$
6 senão se $v_1 < v_2$ então $r \leftarrow \textsc{Ins}(v_1,\textsc{Apply}'(op, t_1, r_2),\textsc{Apply}'(op, f_1, r_2))$
7 senão se $v_1 > v_2$ então $r \leftarrow \textsc{Ins}(v_2,\textsc{Apply}'(op, r_1, t_2),\textsc{Apply}'(op, r_1, f_2))$
8 $M \leftarrow M \cup \{(\langle r_1, r_2\rangle, r)\}$
9 devolva r

7 Implementação em Prolog

```
% bdd.pl (16/Set/2005, Silvio Lago)

:- dynamic t/2, c/1, m/3.

% inicializa a tabela para uma expressao com m variaveis

init(M) :-
    clear(t),
    assert(t(0,[M+1,nil,nil])),
    assert(t(1,[M+1,nil,nil])),
    count(2).

% insere um no N na tabela

ins(N,[_,N,N]) :- !.
ins(N,[V,T,F]) :- t(N,[V,T,F]), !.
ins(N,[V,T,F]) :- count(N), assert(t(N,[V,T,F])).

% constroi um diagrama com raiz R, a partir de uma expressao na forma normal condicional

build(R,ite(V,T,F)) :- member(T,[0,1]), member(F,[0,1]), ins(R,[V,T,F]), !.
build(R,ite(V,T,F)) :- member(F,[0,1]), build(A,T), ins(R,[V,A,F]), !.
build(R,ite(V,T,F)) :- member(T,[0,1]), build(B,F), ins(R,[V,T,B]), !.
build(R,ite(V,T,F)) :- build(A,T), build(B,F), ins(R,[V,A,B]).

% aplica um operador booleano a duas expressoes R1 e R2 e devolve raiz R

apply(Op,R1,R2,R) :-
    clear(m),
    app(Op,R1,R2,R).

app(_,R1,R2,R) :- m(R1,R2,R), !.

app(Op,R1,R2,R) :-
    t(R1,[V1,T1,F1]),
    t(R2,[V2,T2,F2]),
    ((member(R1,[0,1]), member(R2,[0,1])) -> E =.. [Op,R1,R2,R], call(E)
     ; V1=V2 -> app(Op,T1,T2,TR), app(Op,F1,F2,FR), ins(R,[V1,TR,FR])
     ; V1<V2 -> app(Op,T1,R2,TR), app(Op,F1,R2,FR), ins(R,[V1,TR,FR])
     ; V1>V2 -> app(Op,R1,T2,TR), app(Op,R1,F2,FR), ins(R,[V2,TR,FR])),
    assert(m(R1,R2,R)).

% predicados auxiliares

clear(T) :- current_predicate(T,H), retractall(H).
count(N) :- nonvar(N), !, clear(c), assert(c(N)).
count(N) :- retract(c(N)), succ(N,M), assert(c(M)).
```

```
show(D,U) :- member(U,[0,1]), tab(D), format('~w~n',[U]).
show(D,U) :- tab(D), t(U,[V,T,F]), format('x~w~n',[V]),
             succ(D,D1), show(D1,T), show(D1,F).

and(X,Y,Z) :- ((X=1, Y=1) -> Z=1 ; Z=0).
or(X,Y,Z)  :- ((X=0, Y=0) -> Z=0 ; Z=1).

% testes

t1 :- % compartilhamento de subgrafos
    init(4),
    build(R,ite(1,ite(2,ite(3,ite(4,1,0),ite(4,0,1)),0),
                  ite(2,0,ite(3,ite(4,1,0),ite(4,0,1))))),
    show(0,R),
    listing(t).

%   x1
%    x2
%     x3
%      x4
%       1
%       0
%      x4
%       0
%       1
%     0
%    x2
%     0
%     x3
%      x4
%       1
%       0
%      x4
%       0
%       1
%
%   t(0, [4 + 1, nil, nil]).
%   t(1, [4 + 1, nil, nil]).
%   t(2, [4, 1, 0]).
%   t(3, [4, 0, 1]).
%   t(4, [3, 2, 3]).
%   t(5, [2, 4, 0]).
%   t(6, [2, 0, 4]).
%   t(7, [1, 5, 6]).


t2 :- % eliminacao de teste redundante
    init(3),
    build(R,ite(1,ite(2,ite(3,1,0),ite(3,1,0)),0)),
    show(0,R),
    listing(t).

%   x1
%    x3
%     1
%     0
%    0
%
%   t(0, [3 + 1, nil, nil]).
%   t(1, [3 + 1, nil, nil]).
%   t(2, [3, 1, 0]).
%   t(3, [1, 2, 0]).


t3 :- % aplicacao de operador
    init(3),
    build(A,ite(1,0,1)), show(0,A),  get0(_),
    build(B,ite(2,1,0)), show(0,B),  get0(_),
```

```
    build(C,ite(3,1,0)), show(0,C),  get0(_),
    apply(and,A,B,R1),   show(0,R1), get0(_),
    apply(or, R1,C,R2),  show(0,R2), get0(_),
    listing(t).

%    x1
%     0
%     1
%
%    x2
%     1
%     0
%
%    x3
%     1
%     0
%
%    x1
%     0
%     x2
%      1
%      0
%
%    x1
%     x3
%      1
%      0
%     x2
%      1
%      x3
%       1
%       0
%
%   t(0, [3 + 1, nil, nil]).
%   t(1, [3 + 1, nil, nil]).
%   t(2, [1, 0, 1]).
%   t(3, [2, 1, 0]).
%   t(4, [3, 1, 0]).
%   t(5, [1, 0, 3]).
%   t(6, [2, 1, 4]).
%   t(7, [1, 4, 6]).
```

Referências

1. Andersen, H. R. *Introduction to Binary Decision Diagrams*, 1997.
2. Bryant, R. E. *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*, 1992.