Diagramas de Decisão Binária (BDDs)

Aula 2

Luiz Carlos Vieira

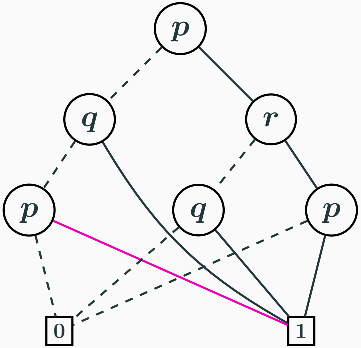
7 de outubro de 2015

MAC0239 - Introdução à Lógica e Verificação de Programas

Conteúdo

- BDDs ordenados e reduzidos (ROBDDs)
- Algoritmos para ROBDDs
  - algoritmo `reduzir`
  - algoritmo `aplicar`
  - algoritmo `restringir`
  - algoritmo `existe`

Relembrando: múltiplas ocorrências

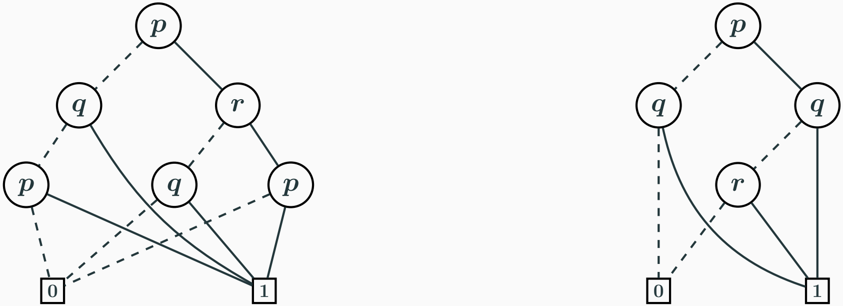


- A definição de BDDs não impede uma variável de ocorrer mais de uma vez em um caminho
- Mas tal representação pode incorrer em desperdícios
  - linha sólida do p à esquerda (colorida) jamais será percorrida

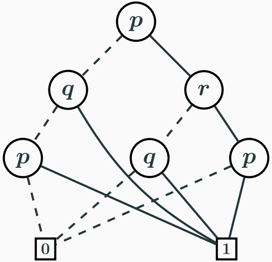
Esse é um resultado comum após as operações discutidas na aula anterior

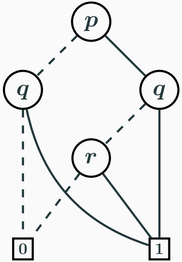
Relembrando: comparação de BDDs

Além de tornar um BDD menos eficiente, ocorrências múltiplas de uma variável também dificultam a comparação de BDDs



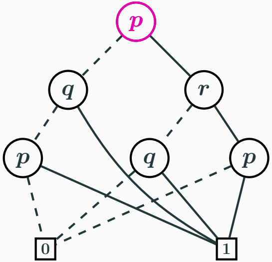
Conceito de ordem de um caminho

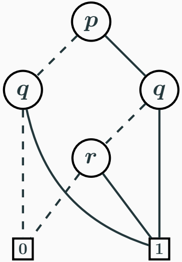




Conceito de ordem de um caminho

- $[p \quad]$



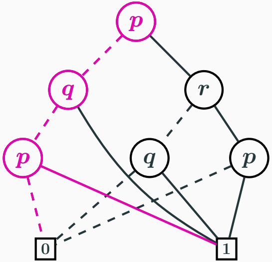


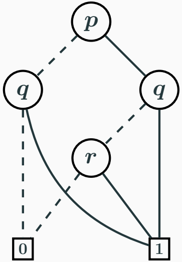
Conceito de ordem de um caminho

- $[p, q \quad]$

Conceito de ordem de um caminho

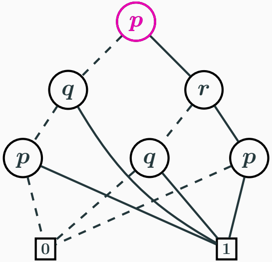
- $[p, q, p]$

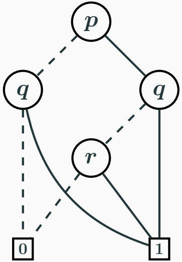




Conceito de ordem de um caminho

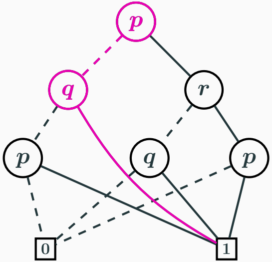
- $[p, q, p]$
- $[p \quad]$

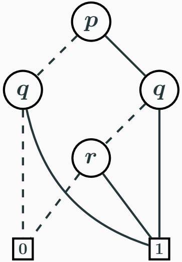




Conceito de ordem de um caminho

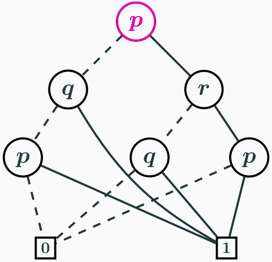
- $[p, q, p]$
- $[p, q]$

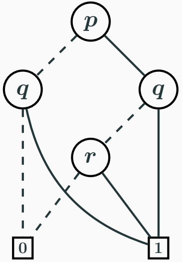




Conceito de ordem de um caminho

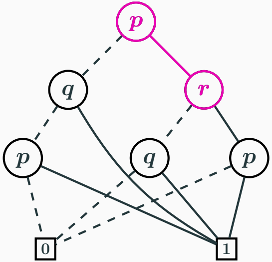
- $[p, q, p]$
- $[p, q]$
- $[p \quad\quad]$

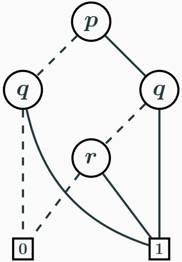




Conceito de ordem de um caminho

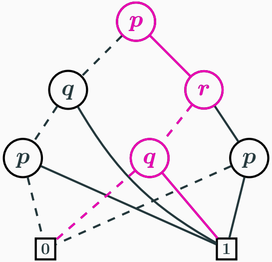
- $[p, q, p]$
- $[p, q]$
- $[p, r \quad]$

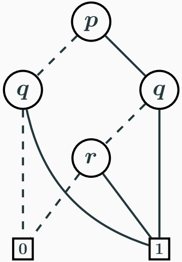




Conceito de ordem de um caminho

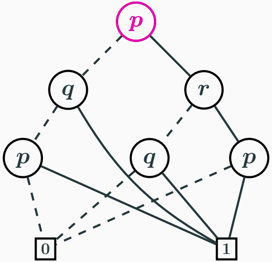
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$

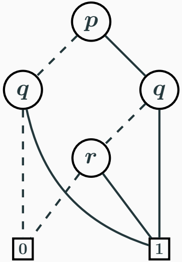




Conceito de ordem de um caminho

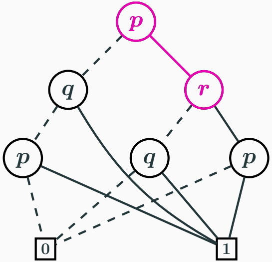
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p \qquad]$

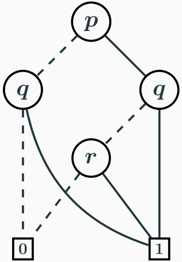




Conceito de ordem de um caminho

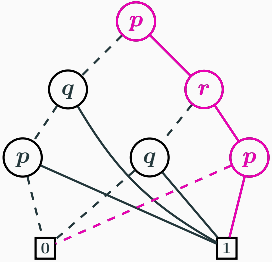
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r \quad]$

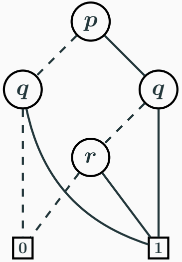




Conceito de ordem de um caminho

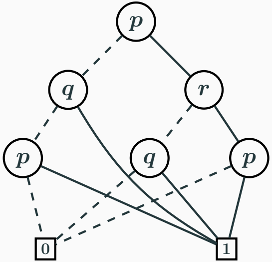
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$



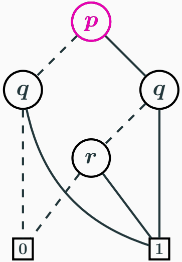


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

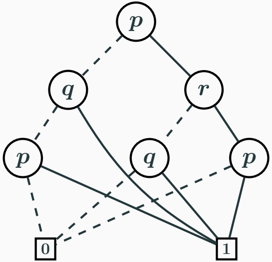


- $[p \quad]$

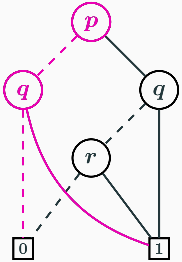


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

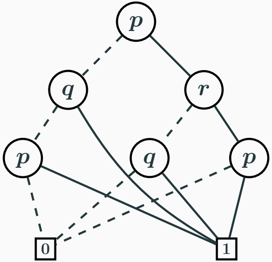


- $[p, q]$

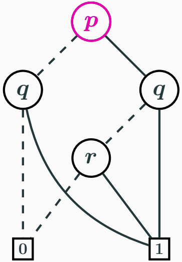


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

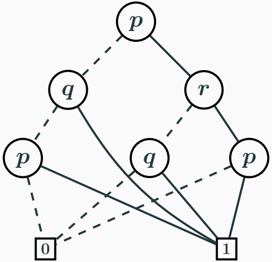


- $[p, q]$
- $[p \quad]$

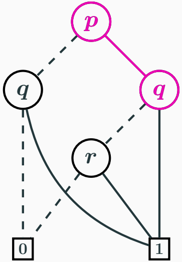


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

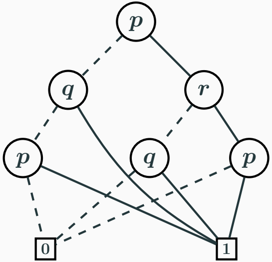


- $[p, q]$
- $[p, q \quad]$

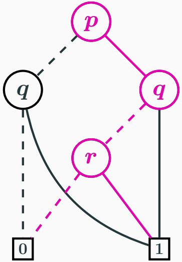


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

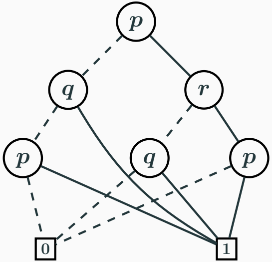


- $[p, q]$
- $[p, q, r]$

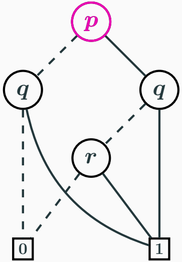


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

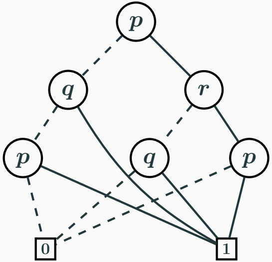


- $[p, q]$
- $[p, q, r]$
- $[p \quad]$

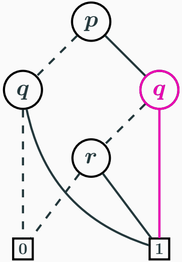


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$

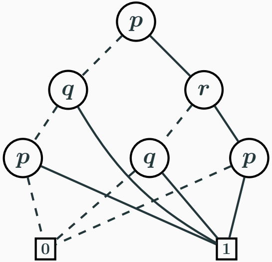


- $[p, q]$
- $[p, q, r]$
- $[p, q]$

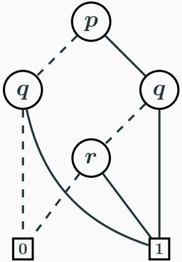


Conceito de ordem de um caminho

- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$



- $[p, q]$
- $[p, q, r]$
- $[p, q]$



BDDs ordenados

Quando a ordem das variáveis em qualquer caminho é sempre a mesma, o BDD passa a ser chamado Diagrama de Busca Binária Ordenado (OBDD)

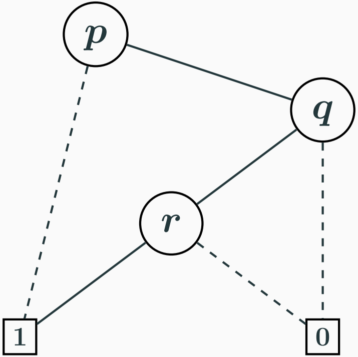
Definição: OBDDs

Definição 6.6

Seja $[\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_n}]$ uma lista ordenada de variáveis <u>sem duplicação</u> e seja $\boldsymbol{B}$ um BDD tal que todas as suas variáveis aparecem em algum lugar da lista. Dizemos que $\boldsymbol{B}$ tem a ordem $[\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_n}]$ se todos os nós de variáveis de $\boldsymbol{B}$ ocorrem na lista, e, para toda ocorrência de $\boldsymbol{p_i}$ seguido de $\boldsymbol{p_j}$ ao longo de qualquer caminho em $\boldsymbol{B}$ (ou seja, $\boldsymbol{p_i} \prec \boldsymbol{p_j}$), temos $\boldsymbol{i} < \boldsymbol{j}$.

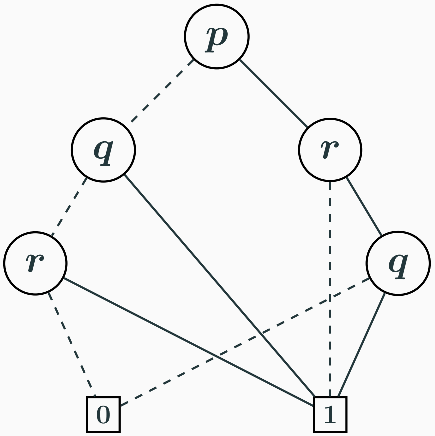
Exemplo de BDD ordenado

Ordem: $[p, q, r]$ (em qualquer caminho)



Exemplo de BDD não ordenado

Sem ordem única ($[p, q, r]$ à esquerda e $[p, r, q]$ à direita)



OBDDs reduzidos

Quando são reduzidos, OBDDs passam a ser chamados de Diagramas de Busca Binária Ordenados Reduzidos (ROBDD)

Vantagens da ordenação de BDDs

- Reduções em um OBDDs mantêm ordem original
  - C1: compartilha nós terminais
  - C2: elimina nós não-terminais redundantes
  - C2: compartilha sub-diagramas idênticos
- Compromisso com ordem e redução produzem representação única de funções booleanas
  - chamada de *forma canônica*
- Comparação de ROBDDs de ordens compatíveis é imediata
  - basta verificar se suas estruturas são idênticas

Teorema: ROBDDs são únicos

Teorema 6.7

A representação em ROBDD de uma função dada ϕ é unica. Isto é, sejam B e B' dois ROBDDs com ordens compatíveis; se B e B' representam a mesma função booleana, então eles têm estruturas idênticas.

Consequências da forma canônica

- **Teste de equivalência semântica**. Se duas funções são representadas por OBDDs com ordens compatíveis, é possível decidir eficientemente se são equivalentes reduzindo-os e comparando sua estrutura;

Consequências da forma canônica

- **Teste de equivalência semântica**. Se duas funções são representadas por OBDDs com ordens compatíveis, é possível decidir eficientemente se são equivalentes reduzindo-os e comparando sua estrutura;
- **Ausência de variáveis redundantes**. Se o valor de uma função booleana não depende de uma variável, então o ROBDD que a representa não contém tal variável;

Consequências da forma canônica

- **Teste de equivalência semântica**. Se duas funções são representadas por OBDDs com ordens compatíveis, é possível decidir eficientemente se são equivalentes reduzindo-os e comparando sua estrutura;
- **Ausência de variáveis redundantes**. Se o valor de uma função booleana não depende de uma variável, então o ROBDD que a representa não contém tal variável;
- **Teste de validade**. Se uma função booleana é válida, seu ROBDD é igual a $\boldsymbol{B_1}$;

Consequências da forma canônica

- **Teste de equivalência semântica**. Se duas funções são representadas por OBDDs com ordens compatíveis, é possível decidir eficientemente se são equivalentes reduzindo-os e comparando sua estrutura;
- **Ausência de variáveis redundantes**. Se o valor de uma função booleana não depende de uma variável, então o ROBDD que a representa não contém tal variável;
- **Teste de validade**. Se uma função booleana é válida, seu ROBDD é igual a $\boldsymbol{B_1}$;
- **Teste de satisfação**. Se uma função booleana é satisfeita, então seu ROBDD não é igual a $\boldsymbol{B_0}$;

Consequências da forma canônica

- **Teste de equivalência semântica**. Se duas funções são representadas por OBDDs com ordens compatíveis, é possível decidir eficientemente se são equivalentes reduzindo-os e comparando sua estrutura;
- **Ausência de variáveis redundantes**. Se o valor de uma função booleana não depende de uma variável, então o ROBDD que a representa não contém tal variável;
- **Teste de validade**. Se uma função booleana é válida, seu ROBDD é igual a $\boldsymbol{B_1}$;
- **Teste de satisfação**. Se uma função booleana é satisfeita, então seu ROBDD não é igual a $\boldsymbol{B_0}$;
- **Teste de implicação**. Pode-se testar se uma função ϕ implica em outra ψ calculando o ROBDD para $\phi \wedge \neg\psi$; a implicação é verdadeira se e somente se este ROBDD é igual a $\boldsymbol{B_0}$.

Impacto da escolha da ordenação

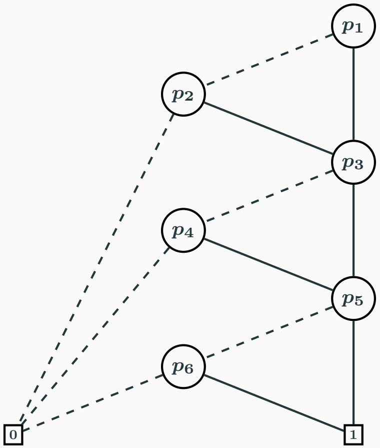
Considere a escolha da ordem de variáveis para a seguinte função booleana em CNF:

$\boldsymbol{\phi \equiv (p_1 \vee p_2) \wedge (p_3 \vee p_4) \wedge \ldots \wedge (p_{2n-1} \vee p_{2n})}$

- Se a escolha for a "*ordem natural de ocorrência na fórmula*" ($\boldsymbol{[p_1, p_2, p_3, \ldots, , p_{2n-1}, p_{2n}]}$), o ROBDD terá $\boldsymbol{2n + 2}$ nós
- Se a escolha for "*índices impares antes de índices pares*" ($\boldsymbol{[p_1, p_3, p_5, \ldots, p_{2n-1}, p_2, p_4, p_6, \ldots, p_{2n}]}$), o ROBDD terá $\boldsymbol{2^{n+1}}$ nós

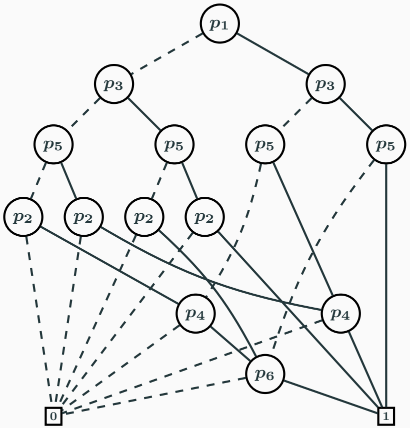
Ordem "natural" para $n = 3$

ROBDD para $\phi \equiv (p_1 \vee p_2) \wedge (p_3 \vee p_4) \wedge (p_5 \wedge p_6)$ com a ordem de variáveis $[p_1, p_2, p_3, p_4, p_5, p_6]$



Ordem "ímpares $\prec$ pares" para $n = 3$

ROBDD para $\phi \equiv (p_1 \vee p_2) \wedge (p_3 \vee p_4) \wedge (p_5 \wedge p_6)$ com a ordem de variáveis $[p_1, p_3, p_5, p_2, p_4, p_6]$



Escolha da ordenação

- A sensibilidade do tamanho de um ROBDD à ordem escolhida é o preço pago pelas facilidades obtidas
- Encontrar a ordem ótima também é um problema computacional caro
  - mas há heurísticas para ordens razoavelmente boas
  - tipicamente, agrupa-se as variáveis com interações mais fortes

ROBDDs como representação

- RODDBs permitem representações compactas de certas classes de funções booleanas
  - que seriam exponenciais em outros formatos/representações
- Por outro lado, não se pode realizar as operações $\wedge$ e $\vee$ da forma anteriormente estudada
  - elas podem introduzir ocorrências múltiplas de variáveis

Operações eficientes com ROBDDs

Utilizando ROBDDs, é possível fazer operações de forma eficiente:

- **Redução**. A redução é o cerne da utilização séria de ROBDDs
  - consistindo da aplicação eficiente das simplificações C1-C3
- **Aplicação**. A aplicação permite realizar as operações lógicas $\wedge$, $\vee$, $\oplus$ e $\neg$ (via $\phi \oplus 1$)
  - mantendo o BDD ordenado e reduzido

Estrutura de dados

Os algoritmos das operações com ROBDDs utilizam como estrutura de dados uma tabela $\boldsymbol{T} : \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle \mapsto \boldsymbol{i_v}$, tal que:

- $\langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_r} \rangle$ representa um nó qualquer no ROBDD
  - com uma variável $\boldsymbol{v}$
  - e identificadores de seus nós-filhos $\boldsymbol{i_l}$ (*low*, pela linha pontilhada) e $\boldsymbol{i_h}$ (*high*, pela linha sólida)
- $\boldsymbol{i_v}$ representa um inteiro positivo que serve como identificador único do nó da variável $\boldsymbol{v}$

Ilustração dessa estrutura de dados

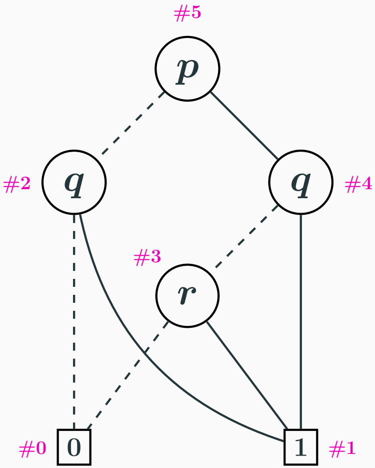


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2
$\langle r, 0, 1 \rangle$	3
$\langle q, 3, 1 \rangle$	4
$\langle p, 2, 4 \rangle$	5

Observações

Nos algoritmos apresentados a seguir, assume-se que:

- A tabela $\boldsymbol{T}$ é uma variável global e $|\boldsymbol{T}|$ é o número de linhas existentes nessa tabela
- $\boldsymbol{T}(\langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle) = \textsc{null}$ quando $(\boldsymbol{i_v}, \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle) \notin \boldsymbol{T}$
- LO e HI acessam os nós-filhos de um nó
- ID acessa o identificador de um nó

Algoritmo de redução

- Percorre os níveis de B de baixo para cima
  - isto é, com busca em profundidade
- Atribui o mesmo identificador a nós que denotam a mesma função
  - compartilhando subdiagramas (simplificações C1/C3)
- Atribui o identificador dos filhos um nó se eles forem os mesmos
  - ignorando nós redundantes (simplificação C2)

Pseudocódigo de GETNODE

precondição: v, i_l, i_h ▷ recebe uma variável e os IDs de seus filhos
pós-condição: i ▷ devolve o ID único do nó da variável

função GETNODE(v, i_l, i_h)
    se $v \neq 0 \land v \neq 1$ **então**
        se $i_l = i_h$ **então** ▷ simplificação C2
            devolva i_l
        fim se
    fim se
    $i \leftarrow T(\langle v, i_l, i_h \rangle)$
    se $i = \mathrm{NULL}$ **então** ▷ simplificações C1/C3
        $i = |T|$
        $T \leftarrow T \cup \{(i, \langle v, i_l, i_h \rangle)\}$
    fim se
    devolva i
fim função

Pseudocódigo de REDUCE

precondição: n ▷ recebe o nó a ser reduzido

pós-condição: i_n ▷ devolve o ID do nó canônico (reduzido)

função $\textsc{reduce}(n)$
  se $n = 0 \lor n = 1$ **então** ▷ simplificação C1
    devolva $\textsc{getnode}(\text{n}, \text{NULL}, \text{NULL})$
  fim se
  $i_n \leftarrow T(\langle n, \textsc{id}(\textsc{lo}(n)), \textsc{id}(\textsc{hi}(n))\rangle)$
  se $i_n = \text{NULL}$ **então**
    $i_l \leftarrow \textsc{reduce}(\textsc{lo}(n))$
    $i_h \leftarrow \textsc{reduce}(\textsc{hi}(n))$
    $i_n = \textsc{getnode}(n, i_l, i_h)$ ▷ simplificação C3
  fim se
  devolva i_n
fim função

Ilustração do algoritmo de redução

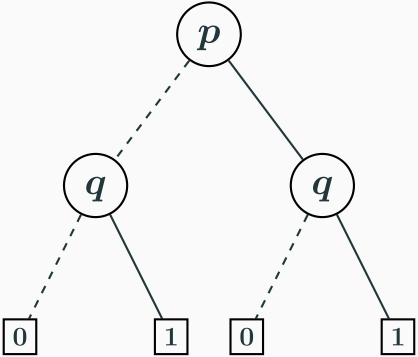


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$

Ilustração do algoritmo de redução

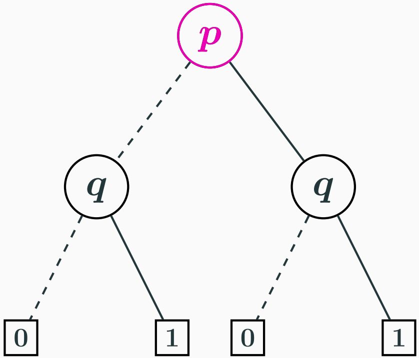


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$

Ilustração do algoritmo de redução

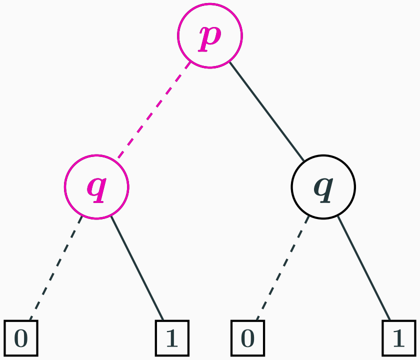


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$

Ilustração do algoritmo de redução

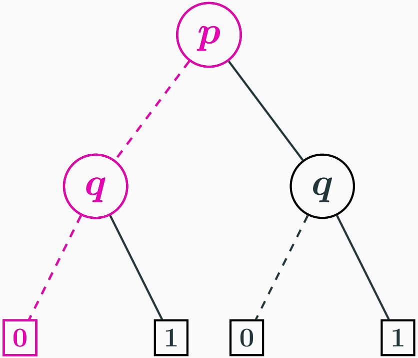


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$

Ilustração do algoritmo de redução

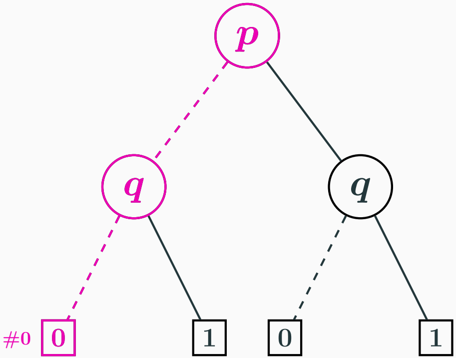


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0

Ilustração do algoritmo de redução

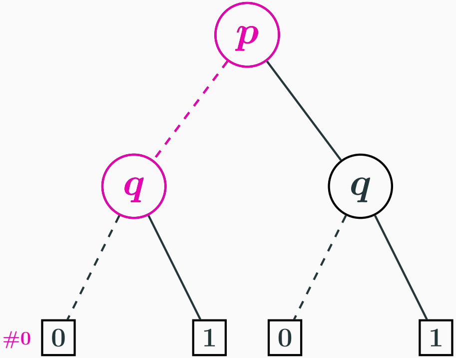


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0

Ilustração do algoritmo de redução

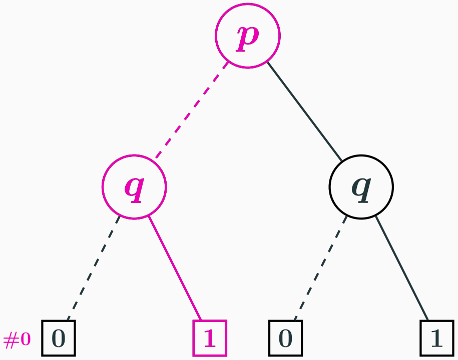


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0

Ilustração do algoritmo de redução

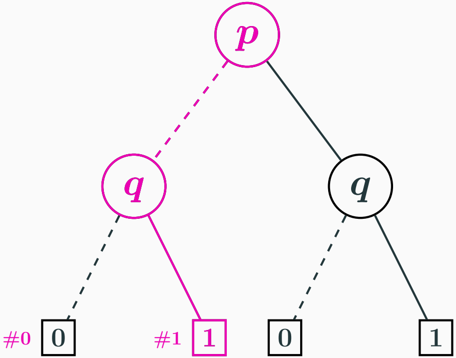


Tabela $\boldsymbol{T : \langle v, i_l, i_h \rangle \mapsto i_v}$

$\boldsymbol{n}$	$\boldsymbol{T(n)}$
$\langle \mathbf{0}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{0}$
$\langle \mathbf{1}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{1}$

Ilustração do algoritmo de redução

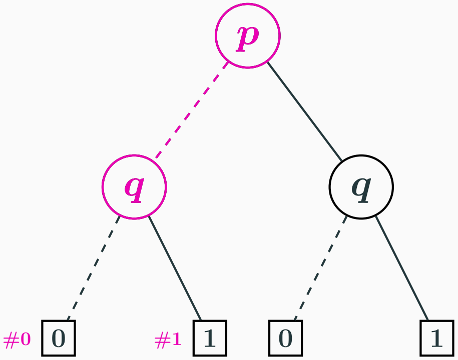


Tabela $\boldsymbol{T} : \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle \mapsto \boldsymbol{i_v}$

$\boldsymbol{n}$	$\boldsymbol{T(n)}$
$\langle \mathbf{0}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{0}$
$\langle \mathbf{1}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{1}$

Ilustração do algoritmo de redução

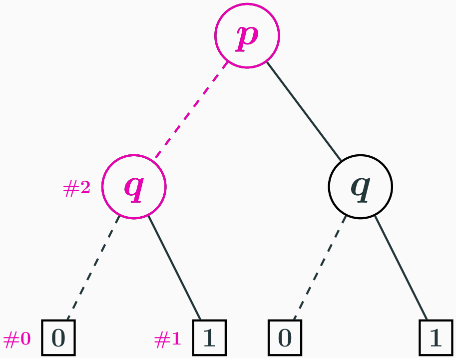


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

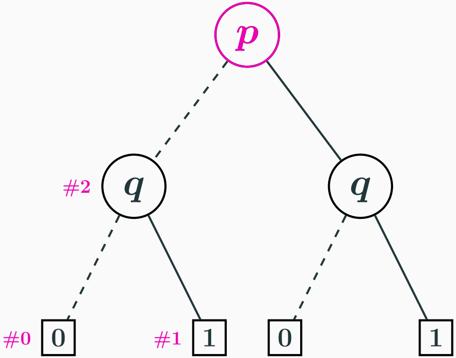


Tabela $\boldsymbol{T} : \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle \mapsto \boldsymbol{i_v}$

$\boldsymbol{n}$	$\boldsymbol{T(n)}$
$\langle \mathbf{0}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{0}$
$\langle \mathbf{1}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{1}$
$\langle \boldsymbol{q}, \mathbf{0}, \mathbf{1} \rangle$	$\mathbf{2}$

Ilustração do algoritmo de redução

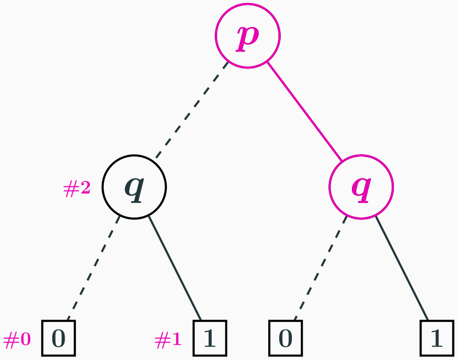


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

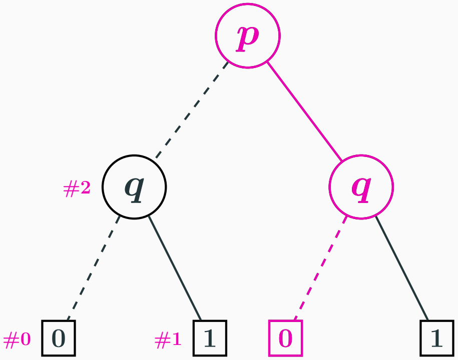


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

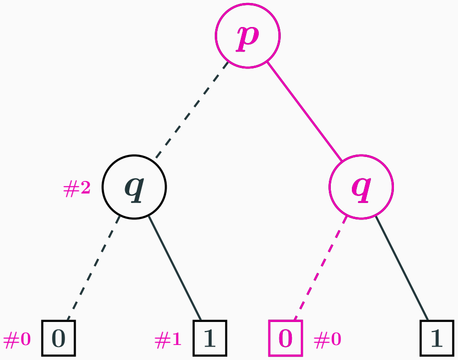


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

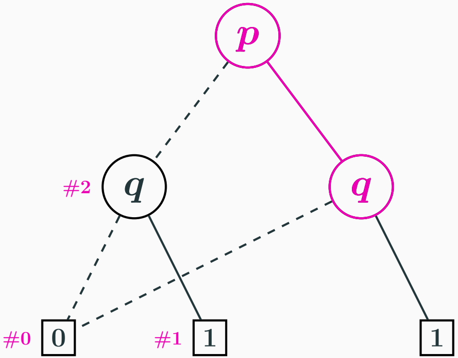


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

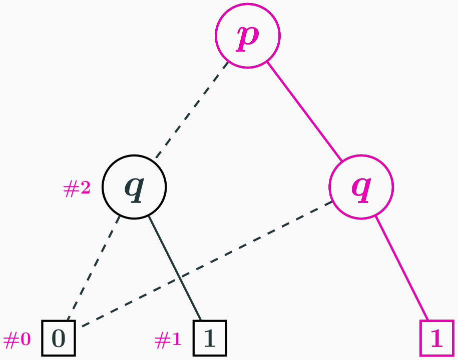


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

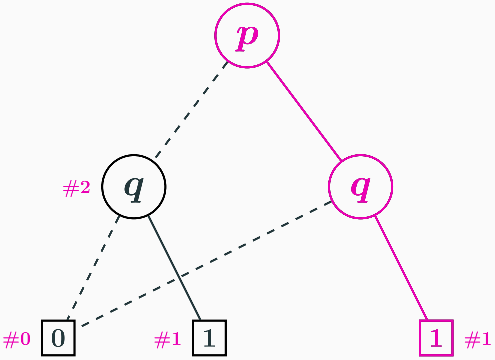


Tabela $\boldsymbol{T : \langle v, i_l, i_h \rangle \mapsto i_v}$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

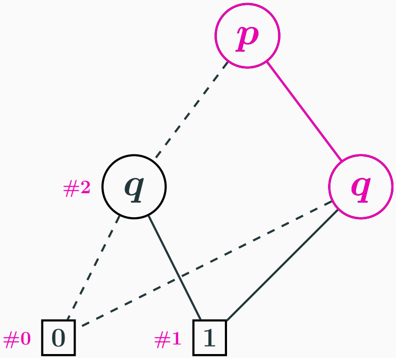


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

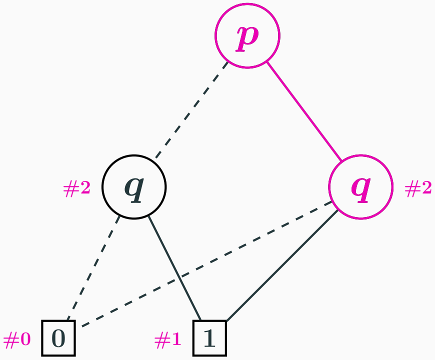


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

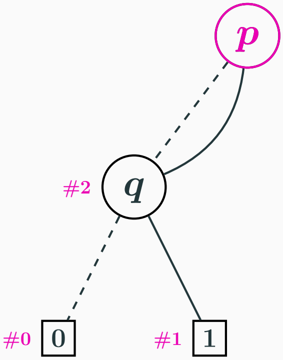


Tabela $\boldsymbol{T} : \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle \mapsto \boldsymbol{i_v}$

$\boldsymbol{n}$	$\boldsymbol{T(n)}$
$\langle \mathbf{0}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{0}$
$\langle \mathbf{1}, \text{NULL}, \text{NULL} \rangle$	$\mathbf{1}$
$\langle \boldsymbol{q}, \mathbf{0}, \mathbf{1} \rangle$	$\mathbf{2}$

Ilustração do algoritmo de redução

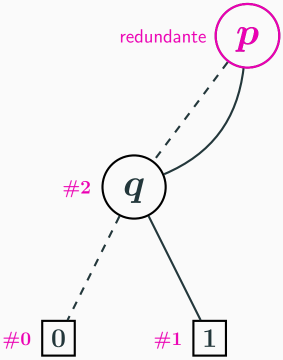


Tabela $T : \langle v, i_l, i_h \rangle \mapsto i_v$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	0
$\langle 1, \text{NULL}, \text{NULL} \rangle$	1
$\langle q, 0, 1 \rangle$	2

Ilustração do algoritmo de redução

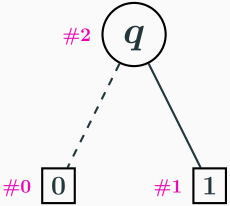


Tabela $\boldsymbol{T} : \langle \boldsymbol{v}, \boldsymbol{i_l}, \boldsymbol{i_h} \rangle \mapsto \boldsymbol{i_v}$

n	$T(n)$
$\langle 0, \text{NULL}, \text{NULL} \rangle$	**0**
$\langle 1, \text{NULL}, \text{NULL} \rangle$	**1**
$\langle q, 0, 1 \rangle$	**2**

Definição: restrições

Definição 6.9

Sejam ϕ uma expressão booleana e p uma variável. Denotamos por $\phi[0/p]$ a expressão booleana obtida substituindo-se todas as ocorrências de p em ϕ por 0. A expressão $\phi[1/p]$ é definida de maneira semelhante. As expressões $\phi[0/p]$ e $\phi[1/p]$ são chamadas de restrições em ϕ com relação à variável p.

Exemplos de restrições

Para $\phi \equiv p \wedge (q \vee \neg p)$ tem-se:

- $\phi[0/p]$ é igual a $0 \wedge (q \vee \neg 0)$
  - que é semanticamente equivalente a 0
- $\phi[1/p]$ é igual a $1 \wedge (q \vee \neg 1)$
  - que é semanticamente equivalente a q
- $\phi[0/q]$ é igual a $p \wedge (0 \vee \neg p)$
  - que é semanticamente equivalente a $\perp$
- $\phi[1/q]$ é igual a $p \wedge (1 \vee \neg p)$
  - que é semanticamente equivalente a p

Uso das restrições

- As restrições permitem executar recorrências em expressões booleanas decompondo-as em expressões mais simples

- Se $\boldsymbol{p}$ é uma variável em ϕ, então ϕ é equivalente a $\neg \boldsymbol{p} \wedge \phi[\mathbf{0}/\boldsymbol{p}] \vee \boldsymbol{p} \wedge \phi[\mathbf{1}/\boldsymbol{p}]$
    - facilmente verificável
    - fazendo $\boldsymbol{p} = \mathbf{0}$ resulta em $\phi[\mathbf{0}/\boldsymbol{p}]$
    - fazendo $\boldsymbol{p} = \mathbf{1}$ resulta em $\phi[\mathbf{1}/\boldsymbol{p}]$

Lema: Expansão de Shannon

Lema 6.10

Para todas as expressões booleanas ϕ e todas as variáveis p (mesmo as que não ocorrem em ϕ), tem-se a chamada Expansão de Shannon:

$$\phi \equiv \neg p \wedge \phi[0/p] \vee p \wedge \phi[1/p]$$