

# Diagramas de Decisão Binária (BDDs)

## Aula 2

---

Luiz Carlos Vieira

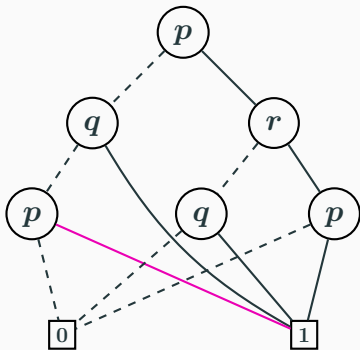
7 de outubro de 2015

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

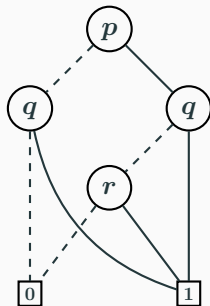
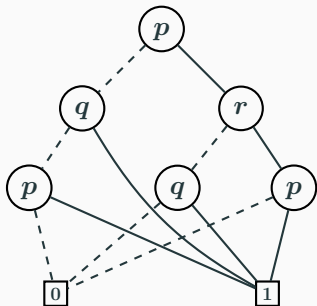


- A definição de BDDs não impede uma variável de ocorrer mais de uma vez em um caminho
- Mas tal representação pode incorrer em desperdícios
  - linha sólida do  $p$  à esquerda (colorida) jamais será percorrida

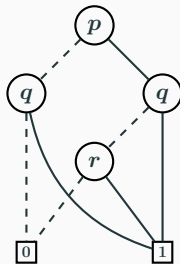
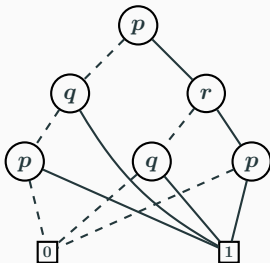
Esse é um resultado comum após as operações discutidas na aula anterior

# Relembrando: comparação de BDDs

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

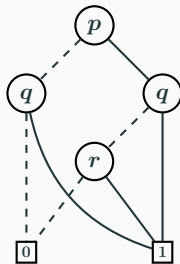
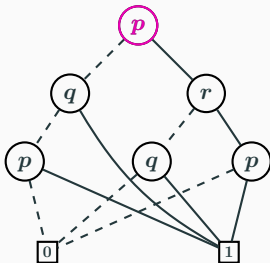


# Conceito de ordem de um caminho



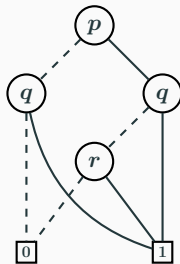
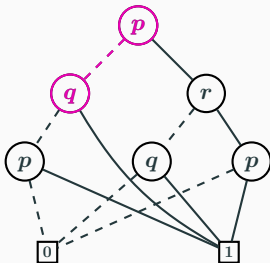
# Conceito de ordem de um caminho

- $[p \quad ]$



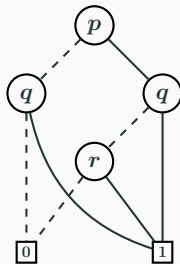
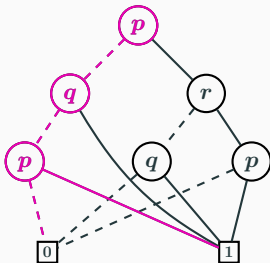
# Conceito de ordem de um caminho

- $[p, q]$



# Conceito de ordem de um caminho

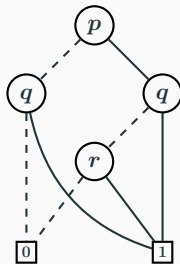
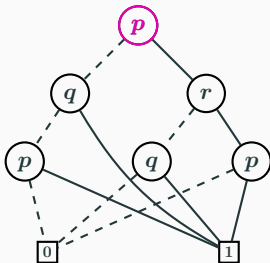
- $[p, q, p]$





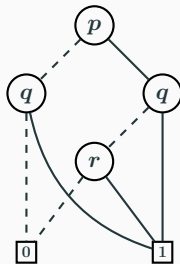
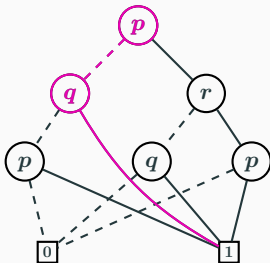
# Conceito de ordem de um caminho

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



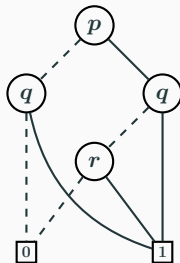
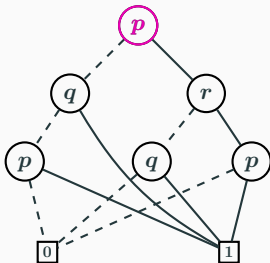
# Conceito de ordem de um caminho

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



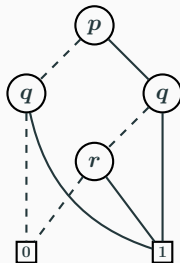
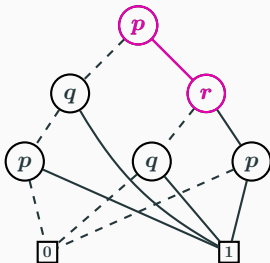
# Conceito de ordem de um caminho

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



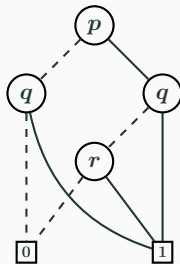
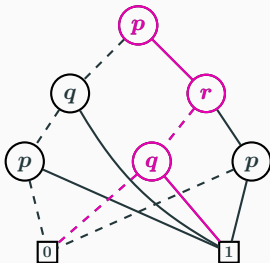
# Conceito de ordem de um caminho

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



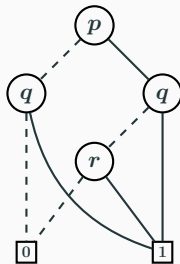
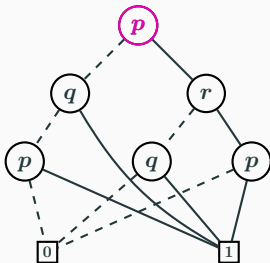
# Conceito de ordem de um caminho

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



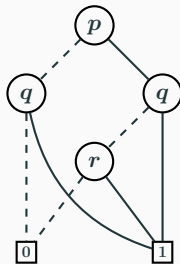
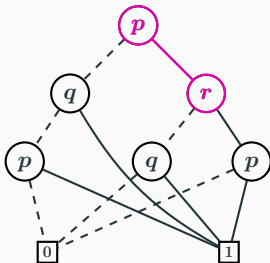
# Conceito de ordem de um caminho

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



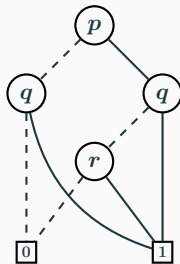
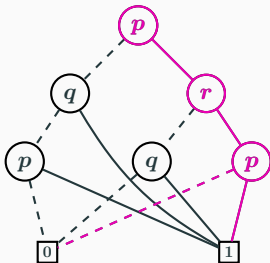
# Conceito de ordem de um caminho

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



# Conceito de ordem de um caminho

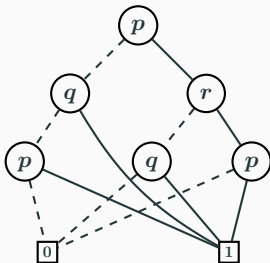
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$



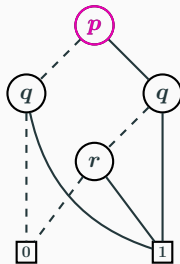


# Conceito de ordem de um caminho

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

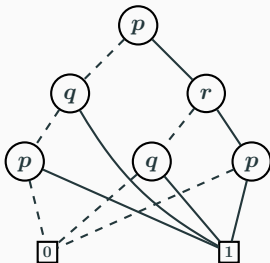


- $[p]$

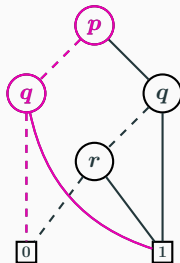


# Conceito de ordem de um caminho

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

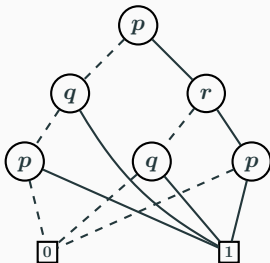


- $[p, q]$

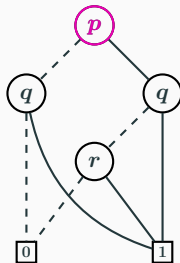


# Conceito de ordem de um caminho

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

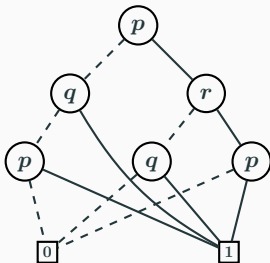


- $[p, q]$
- $[p$  ]

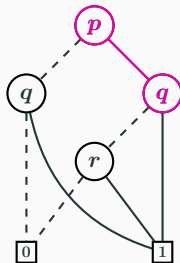


# Conceito de ordem de um caminho

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

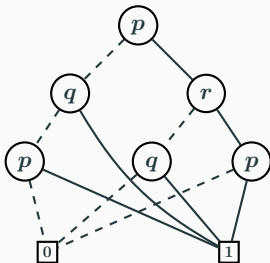


- $[p, q]$
- $[p, q]$

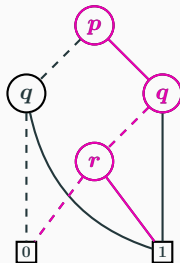


# Conceito de ordem de um caminho

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

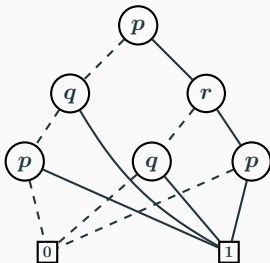


- $[p, q]$
- $[p, q, r]$

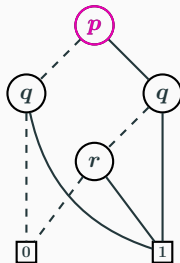


# Conceito de ordem de um caminho

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

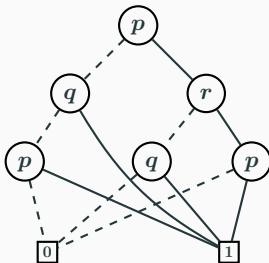


- $[p, q]$
- $[p, q, r]$
- $[p]$

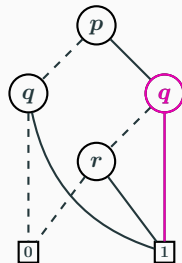


# Conceito de ordem de um caminho

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

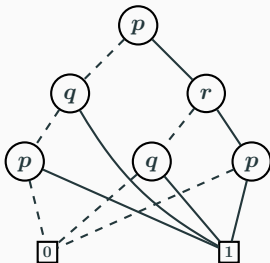


- $[p, q]$
- $[p, q, r]$
- $[p, q]$

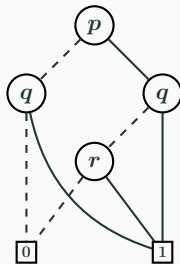


# Conceito de ordem de um caminho

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



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





# 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)

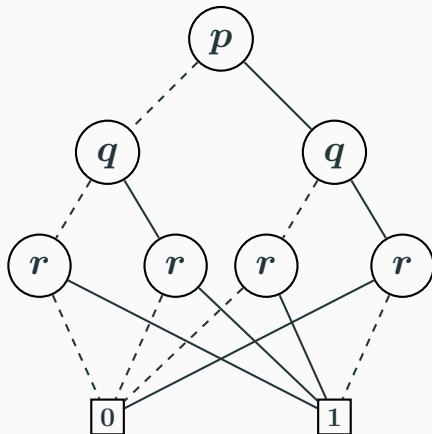
# Definição: OBDDs

## Definição 6.6

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

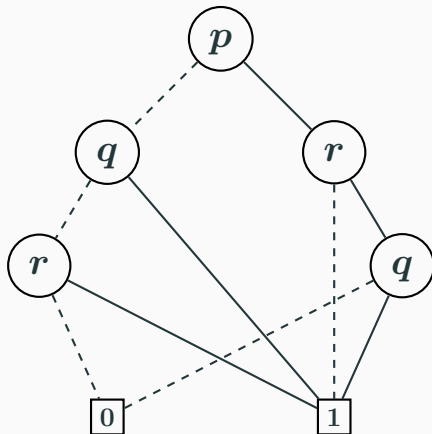
# Exemplo de BDD ordenado

Ordem:  $[p, q, r]$



# Exemplo de BDD não ordenado

Não ordenado ( $[p, q, r]$  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

- Aplicações das reduções C1-C3 em um OBDD garantidamente mantêm a ordem original
- O compromisso com a ordem e o processo de redução produzem uma representação única de funções booleanas
  - chamada de *forma canônica*
- A comparação de dois 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  $\phi$  é única. 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.

# Características de ROBDDs

- 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
  - pois introduzem ocorrências múltiplas de uma mesma variável



# Impacto da escolha da ordenação

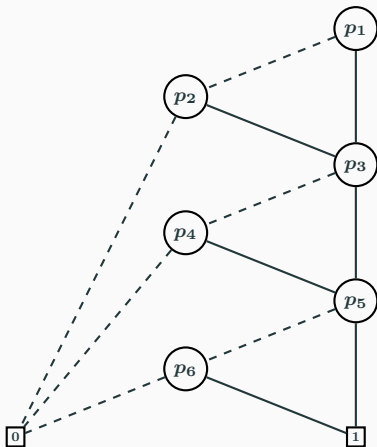
Considere a escolha da ordem de variáveis para a seguinte função booleana em CNF:

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

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

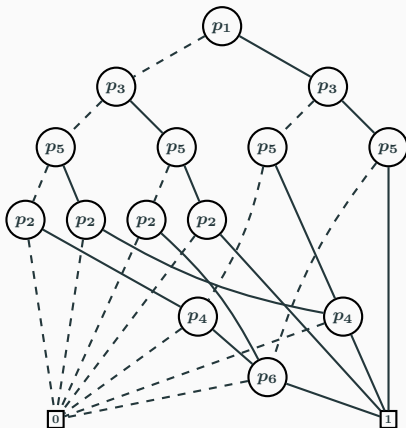
# Ordem “natural” para $n = 3$

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 “ímpar/par” 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 um preço que se paga pelas vantagens obtidas
- Encontrar a ordem ótima também é um problema computacional caro
  - mas há heurísticas\* que produzem ordens razoavelmente boas

\* tipicamente agrupando as variáveis com interações mais fortes

# 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 seus OBDDs 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 seus OBDDs e comparando sua estrutura;
- **Ausência de variáveis redundantes.** Se o valor de uma função booleana não depende de uma variável, então nenhum ROBDD que a represente 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 seus OBDDs e comparando sua estrutura;
- **Ausência de variáveis redundantes.** Se o valor de uma função booleana não depende de uma variável, então nenhum ROBDD que a represente contém tal variável;
- **Teste de validade.** Se uma função booleana é válida, seu ROBDD é igual a  $B_1$ ;

# 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 seus OBDDs e comparando sua estrutura;
- **Ausência de variáveis redundantes.** Se o valor de uma função booleana não depende de uma variável, então nenhum ROBDD que a represente contém tal variável;
- **Teste de validade.** Se uma função booleana é válida, seu ROBDD é igual a  $B_1$ ;
- **Teste de satisfação.** Se uma função booleana é satisfeita, então seu ROBDD não é igual a  $B_0$ ;



# 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 seus OBDDs e comparando sua estrutura;
- **Ausência de variáveis redundantes.** Se o valor de uma função booleana não depende de uma variável, então nenhum ROBDD que a represente contém tal variável;
- **Teste de validade.** Se uma função booleana é válida, seu ROBDD é igual a  $B_1$ ;
- **Teste de satisfação.** Se uma função booleana é satisfeita, então seu ROBDD não é igual a  $B_0$ ;
- **Teste de implicação.** Pode-se testar se uma função  $\phi$  implica em outra  $\psi$  calculando o ROBDD para  $\phi \wedge \neg\psi$ ; a implicação é verdadeira se e somente este ROBDD é igual a  $B_0$ .

# Antes de prosseguir...

Antes de estudarmos as operações sobre ROBDDs, é necessário estudar um conceito importantíssimo: a *expansão de Shannon*

# Definição: restrições

## Definição 6.9

Sejam  $\phi$  uma expressão booleana e  $p$  uma variável. Denotamos por  $\phi[0/p]$  a expressão booleana obtida substituindo-se todas as ocorrências de  $p$  em  $\phi$  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  $\phi$  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  $p$  é uma variável em  $\phi$ , então  $\phi$  é equivalente a  $\neg p \wedge \phi[0/p] \vee p \wedge \phi[1/p]$ 
  - facilmente verificável
  - fazendo  $p = 0$  resulta em  $\phi[0/p]$
  - fazendo  $p = 1$  resulta em  $\phi[1/p]$

# Lema: expansão de Shannon

## Lema 6.10

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

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

# operador ITE

## Definição auxiliar<sup>1</sup>

Com base nessa equivalência, definimos o operador ITE (de *if-then-else*) como:

$$\text{ITE}(p, \phi, \phi') = (p \wedge \phi) \vee (\neg p \wedge \phi')$$

Dessa forma, ITE é verdadeiro se a variável  $p$  e a expressão  $\phi$  são verdadeiros, ou se a variável  $p$  é falsa e a expressão  $\phi'$  é verdadeira.

---

<sup>1</sup>não consta do livro

# Uso do operador ITE

O operador  $\text{ITE}(p, \phi, \phi') = (p \wedge \phi) \vee (\neg p \wedge \phi')$  permite expressar qualquer operador da gramática da Lógica Proposicional:

- $\neg p = \text{ITE}(p, \neg 1, \neg 0) = \text{ITE}(p, 0, 1)$
- $p \wedge q = \text{ITE}(p, 1 \wedge q, 0 \wedge q) = \text{ITE}(p, q, 0) = \text{ITE}(p, \text{ITE}(q, 1, 0), 0)$
- $p \vee q = \text{ITE}(p, 1 \vee q, 0 \vee q) = \text{ITE}(p, 1, q) = \text{ITE}(p, 1, \text{ITE}(q, 1, 0))$
- $p \rightarrow q = \text{ITE}(p, 1 \rightarrow q, 0 \rightarrow q) = \text{ITE}(p, q, 1) = \text{ITE}(p, \text{ITE}(q, 1, 0), 1)$



# Forma normal condicional

Uma expressão booleana está na forma normal condicional se e somente se ela contém apenas constantes, o operador condicional ITE e variáveis sendo testadas por esse operador

# Exemplos na forma e fora dela

- A expressão  $\text{ITE}(p, \text{ITE}(q, 1, r), \text{ITE}(q, 1, 0))$  não está na forma normal condicional
  - a variável  $r$  ainda não está sendo testada condicionalmente
- A expressão  $\text{ITE}(p, \text{ITE}(q, 1, \text{ITE}(r, 1, 0)), \text{ITE}(q, 1, 0))$  está na forma normal condicional
  - todas as variáveis estão sendo testadas condicionalmente

# Conversão para a forma condicional

Toda expressão booleana pode ser convertida indutivamente para a forma normal condicional da seguinte maneira:

- Se  $\phi$  só contém variáveis de teste, ela já está na forma normal condicional
- Senão, enquanto houver uma variável  $p \in \phi$  que não seja teste, reescreva  $\phi$  como  $\text{ITE}(p, \phi[1/p], \phi[0/p])$

# Por exemplo

Conversão de  $(p \vee q) \wedge (q \vee r)$  para a forma normal condicional

$$\begin{aligned} &= \text{ITE}(\textcolor{violet}{p}, (\textcolor{violet}{1} \vee q) \wedge (q \vee r), (\textcolor{violet}{0} \vee q) \wedge (q \vee r)) \\ &= \text{ITE}(p, q \vee r, q) \\ &= \text{ITE}(p, \text{ITE}(\textcolor{violet}{q}, \textcolor{violet}{1} \vee r, \textcolor{violet}{0} \vee r), \text{ITE}(\textcolor{violet}{q}, \textcolor{violet}{1}, \textcolor{violet}{0})) \\ &= \text{ITE}(p, \text{ITE}(q, 1, r), \text{ITE}(q, 1, 0)) \\ &= \text{ITE}(p, \text{ITE}(q, 1, \text{ITE}(\textcolor{violet}{r}, \textcolor{violet}{1}, \textcolor{violet}{0})), \text{ITE}(q, 1, 0)) \end{aligned}$$

# Forma condicional e árvore de decisão

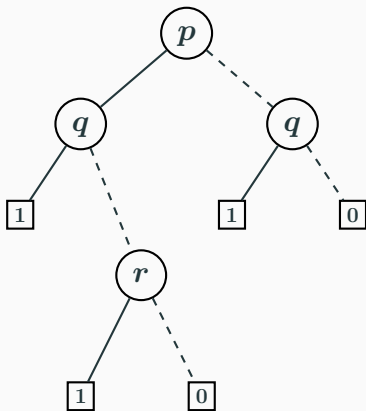
A estrutura de recorrência da forma normal condicional:

$$\text{ITE}(p, \text{ITE}(q, 1, \text{ITE}(r, 1, 0)), \text{ITE}(q, 1, 0))$$

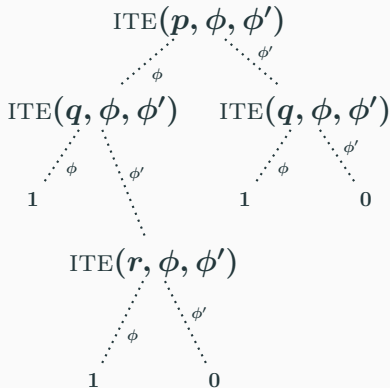
da expressão booleana  $(p \vee q) \wedge (q \vee r)$  é a mesma da sua árvore de decisão binária

# Ilustração do argumento anterior

Árvore de Decisão Binária da expressão:  
 $(p \vee q) \wedge (q \vee r)$



Estrutura recorrente da forma condicional:  
 $\text{ITE}(p, \text{ITE}(q, 1, \text{ITE}(r, 1, 0)), \text{ITE}(q, 1, 0))$



# Estrutura de dados

A estrutura de dados para representar um ROBDD é composta de:

- 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  $f$
- Uma tabela inversa  $T^{-1} : \langle v, t, f \rangle \mapsto n$ 
  - que associa nós em identificadores
  - devido ao compartilhamento de sub-grafos
  - usada para garantir que os diagramas sejam reduzidos

# Ilustração dessa estrutura de dados

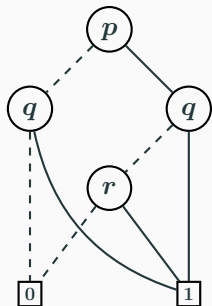


Tabela  $T : n \mapsto \langle v, t, f \rangle$

$n$	$T(n)$
0	$\langle p_6, \text{NULL}, \text{NULL} \rangle$
1	$\langle p_6, \text{NULL}, \text{NULL} \rangle$
2	$\langle p, 3, 4 \rangle$
3	$\langle q, 0, 1 \rangle$
4	$\langle q, 5, 1 \rangle$
5	$\langle r, 0, 1 \rangle$

Tabela  $T^{-1} : \langle v, t, f \rangle \mapsto n$

$\langle v, t, f \rangle$	$T^{-1}(\langle v, t, f \rangle)$
$\langle p_6, \text{NULL}, \text{NULL} \rangle$	0
$\langle p_6, \text{NULL}, \text{NULL} \rangle$	1
$\langle p, 3, 4 \rangle$	2
$\langle q, 0, 1 \rangle$	3
$\langle q, 5, 1 \rangle$	4
$\langle r, 0, 1 \rangle$	5

$p_6$ : variável auxiliar usada nos nós terminais para manter a uniformidade da tabela



# Observações

Nos algoritmos estudados a seguir, assume-se que:

- $T(n) = T^{-1}(\langle v, t, f \rangle) = \text{NULL}$  sempre que  $(n, \langle v, t, f \rangle) \notin T$
- A tabela  $T$  é uma variável global e  $|T|$  é o número de entradas existentes nessa tabela

# Algoritmo de inicialização

Cria a tabela  $T$  de um ROBDD. Funciona assim:

- Recebe uma entrada  $m$  indicando o número máximo de variáveis existentes na expressão booleana
- Inicia a tabela  $T$  com duas tuplas especiais
  - representando os nós terminais  $0$  e  $1$
  - para garantir uniformidade, associa os nós terminais à uma variável auxiliar  $p_{m+1}$

# Pseudocódigo de INIT

---

---

```
1: procedure INIT( $T, m$ )
2:    $T \leftarrow \{(0, \langle m + 1, \text{NULL}, \text{NULL} \rangle)\},$ 
       $\{(1, \langle m + 1, \text{NULL}, \text{NULL} \rangle)\}$ 
```

---

# Algoritmo de inserção de nós

Inserir um nó em um ROBDD, mantendo-o reduzido e ordenado. Funciona assim:

- Recebe como entrada uma variável  $v$  e os identificadores de seus filhos  $t$  e  $f$
- Se o nó  $v$  for redundante ( $t = f$ ), devolve imediatamente o identificador do nó filho ( $t$ )
- Caso o nó  $v$  já tenha sido criado, devolve seu identificador
- Caso o nó  $v$  seja novo, cria-o e devolve o identificador

# Pseudocódigo de INS

---

---

```
1: function INS( $T, v, t, f$ )
2:   if  $t = f$  then
3:     return  $t$ 
4:    $n \leftarrow T^{-1}(\langle v, t, f \rangle)$ 
5:   if  $n = \text{NULL}$  then
6:      $n \leftarrow |T|$ 
7:      $T \leftarrow T \cup \{(n, \langle v, t, f \rangle)\}$ 
8:   return  $n$ 
```

---

# Algoritmo de construção de ROBDDs

Constrói um ROBDD a partir de uma expressão em forma normal condicional. Funciona assim:

- Recebe como entrada uma variável  $v$  e os identificadores de seus filhos  $t$  e  $f$
- Se o nó  $v$  for redundante ( $t = f$ ), devolve imediatamente o identificador do nó filho ( $t$ )
- Caso o nó  $v$  já tenha sido criado, devolve seu identificador
- Caso o nó  $v$  seja novo, cria-o e devolve o identificador

# Pseudocódigo de BUILD

---

```
1: function BUILD(ITE( $v, \phi_t, \phi_f$ ))
2:   if  $\phi_t, \phi_f \in \{0, 1\}$  then
3:     return INS( $v, \phi_t, \phi_f$ )
4:   if  $\phi_f \in \{0, 1\}$  then
5:     return INS( $v$ , BUILD( $\phi_t$ ),  $\phi_f$ )
6:   if  $\phi_t \in \{0, 1\}$  then
7:     return INS( $v, \phi_t$ , BUILD( $\phi_f$ ))
8:   return INS( $v$ , BUILD( $\phi_t$ ), BUILD( $\phi_f$ ))
```

---