Diagramas de Decisão Binária (BDDs)

Aula 2

Luiz Carlos Vieira

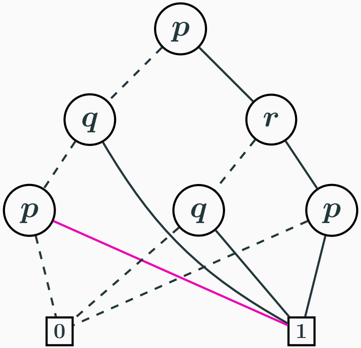
7 de outubro de 2015

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



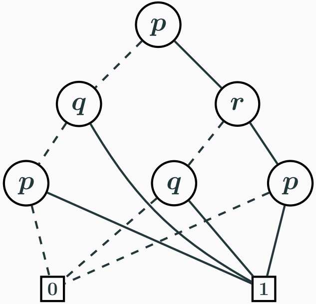
- A definição de BDDs não impede uma variável de ocorrer mais de uma vez em um caminho

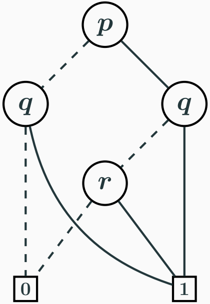
- Mas tal representação pode incorrer em desperdícios
  - linha sólida do p à esquerda (colorida) jamais será percorrida

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

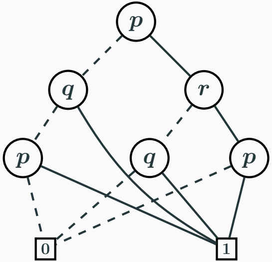
Relembrando: comparação de BDDs

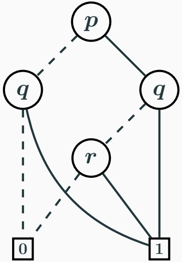
Além de tornar um BDD menos eficiente, ocorrências múltiplas de uma variável também dificultam a comparação de BDDs





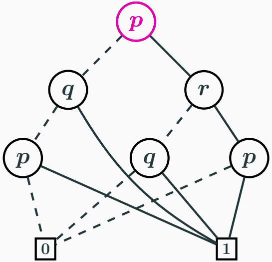
Conceito de ordem de um caminho

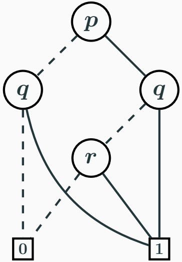




Conceito de ordem de um caminho

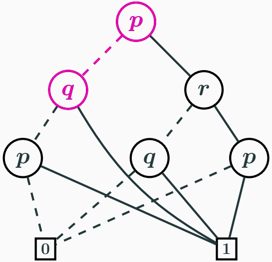
- $[p \qquad]$

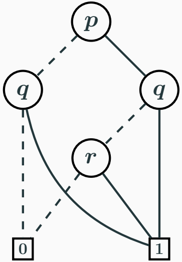




Conceito de ordem de um caminho

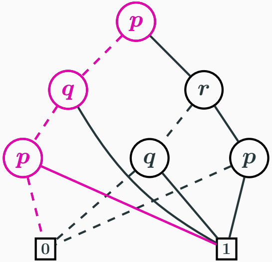
- $[p, q \quad]$

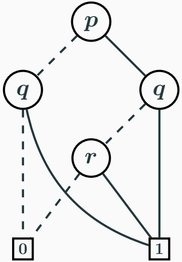




Conceito de ordem de um caminho

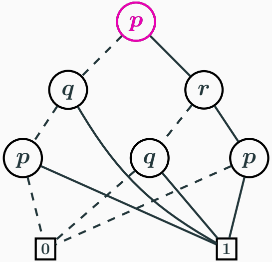
- $[p, q, p]$

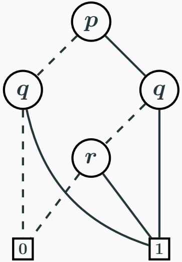




Conceito de ordem de um caminho

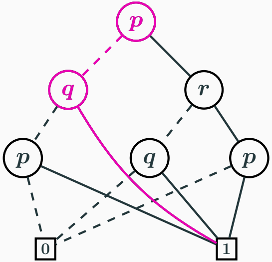
- $[p, q, p]$
- $[p \quad]$

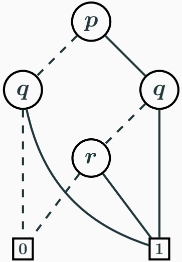




Conceito de ordem de um caminho

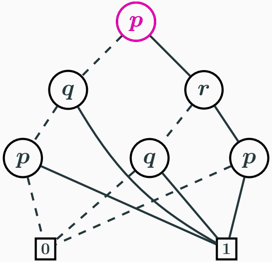
- $[p, q, p]$
- $[p, q]$

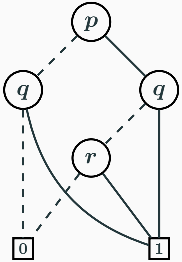




Conceito de ordem de um caminho

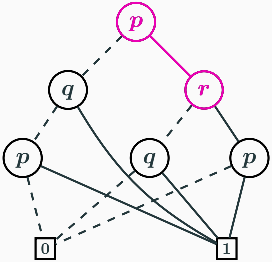
- $[p, q, p]$
- $[p, q]$
- $[p \quad\quad]$

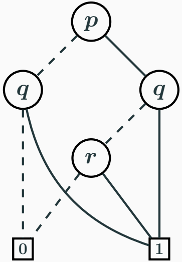




Conceito de ordem de um caminho

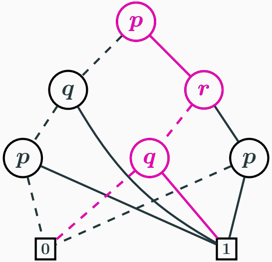
- $[p, q, p]$
- $[p, q]$
- $[p, r \quad]$

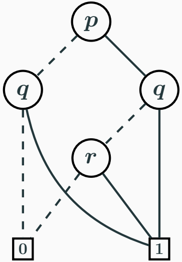




Conceito de ordem de um caminho

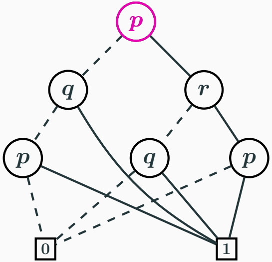
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$

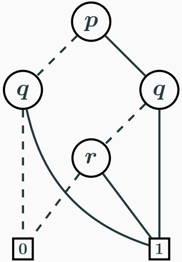




Conceito de ordem de um caminho

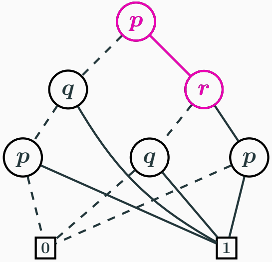
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p \quad\quad]$

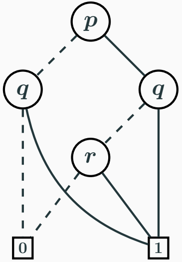




Conceito de ordem de um caminho

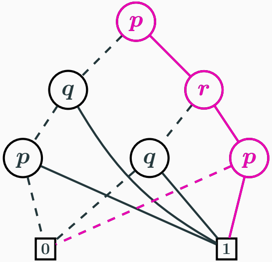
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r \quad]$

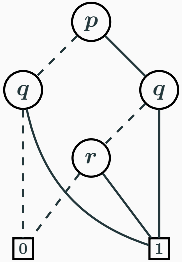




Conceito de ordem de um caminho

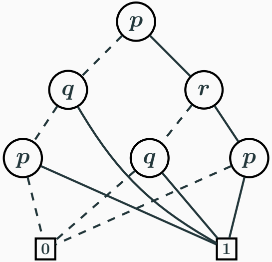
- $[p, q, p]$
- $[p, q]$
- $[p, r, q]$
- $[p, r, p]$



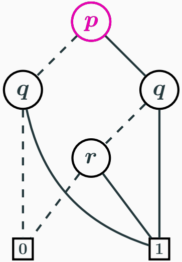


Conceito de ordem de um caminho

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

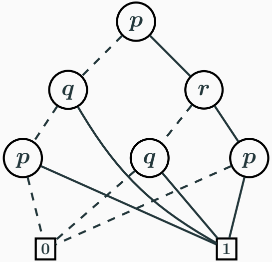


- $[p \quad]$

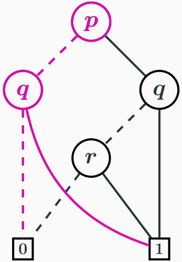


Conceito de ordem de um caminho

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

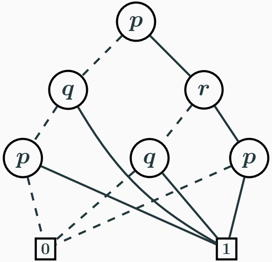


- $[p, q]$

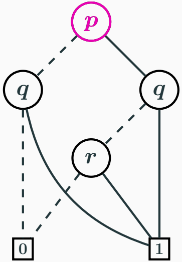


Conceito de ordem de um caminho

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

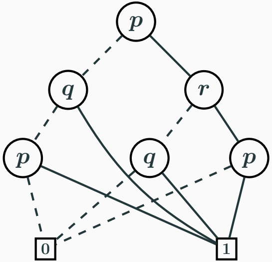


- $[p, q]$
- $[p \quad\quad]$

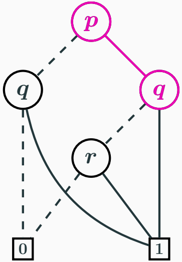


Conceito de ordem de um caminho

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

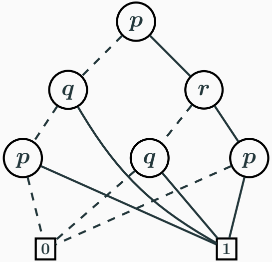


- $[p, q]$
- $[p, q \quad]$

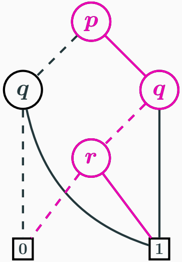


Conceito de ordem de um caminho

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

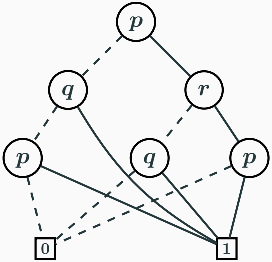


- $[p, q]$
- $[p, q, r]$

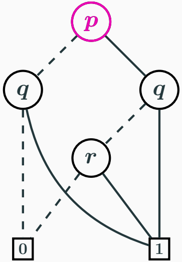


Conceito de ordem de um caminho

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

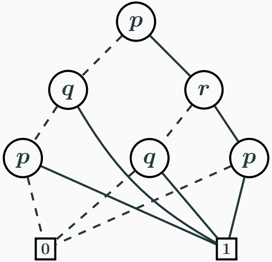


- $[p, q]$
- $[p, q, r]$
- $[p \quad]$

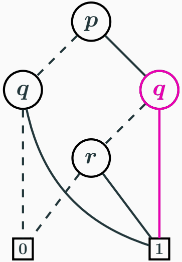


Conceito de ordem de um caminho

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

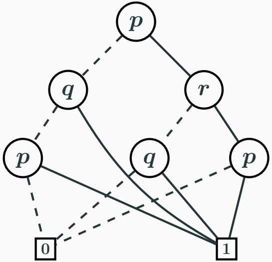


- $[p, q]$
- $[p, q, r]$
- $[p, q]$

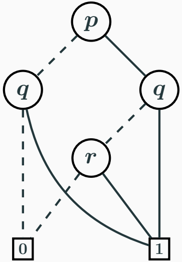


Conceito de ordem de um caminho

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



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



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)

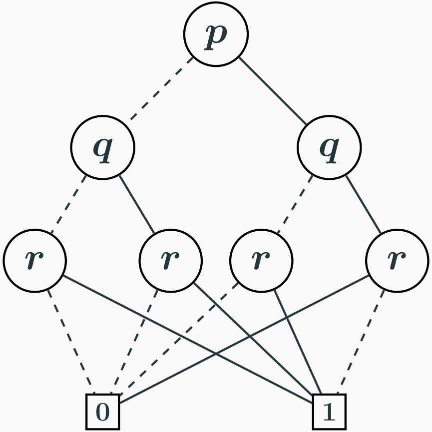
Definição: OBDDs

Definição 6.6

Seja $[p_1, p_2, ..., p_n]$ 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 $[p_1, p_2, ..., p_n]$ 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 $i < j$.

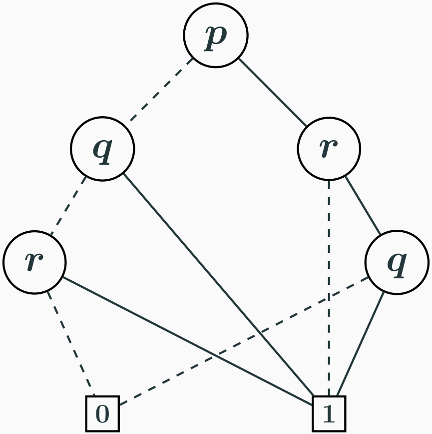
Exemplo de BDD ordenado

Ordem: $[\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}]$



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 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.

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 $\land$ e $\lor$ da forma anteriormente estudada
  - pois introduzem ocorrências múltiplas de uma mesma variável

Impacto da escolha da ordenação

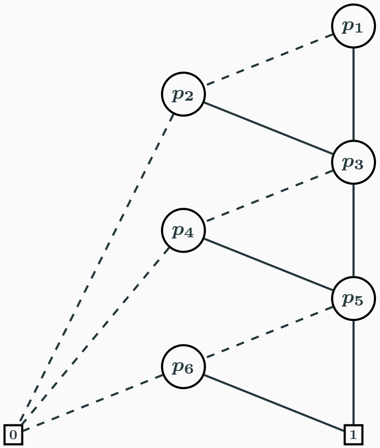
Considere a escolha da ordem de variáveis para a seguinte função booleana em CNF:

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

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

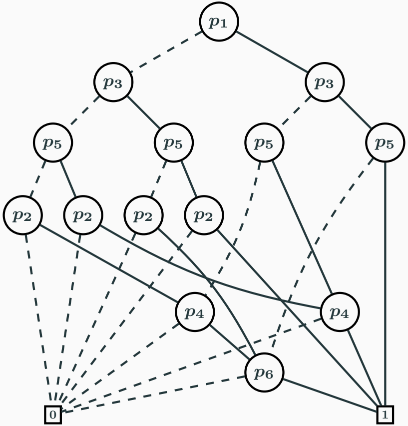
Ordem "natural" para $n = 3$

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 $\boldsymbol{B_1}$;

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 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 $\boldsymbol{B_1}$;
- **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 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 $\boldsymbol{B_1}$;

- **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 $\phi \land \neg\psi$; a implicação é verdadeira se e somente este ROBDD é igual a $\boldsymbol{B_0}$.

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 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 p é uma variável em ϕ, então ϕ 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 ϕ e todas as variáveis p (mesmo as que não ocorrem em ϕ), tem-se a chamada expansão de Shannon:

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

operador ITE

Definição auxiliar[1]

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

$$\textsc{ite}(\boldsymbol{p}, \boldsymbol{\phi}, \boldsymbol{\phi'}) = (\boldsymbol{p} \wedge \boldsymbol{\phi}) \vee (\neg \boldsymbol{p} \wedge \boldsymbol{\phi'})$$

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

[1] 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 $\text{ITE}(\boldsymbol{p}, \text{ITE}(\boldsymbol{q}, 1, \boldsymbol{r}), \text{ITE}(\boldsymbol{q}, 1, 0))$ não está na forma normal condicional
  - a variável $\boldsymbol{r}$ ainda não está sendo testada condicionalmente

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

Por exemplo

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

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

Forma condicional e árvore de decisão

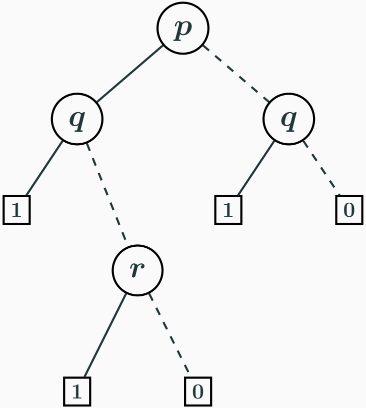
A estrutura de recorrência da forma normal condicional:

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

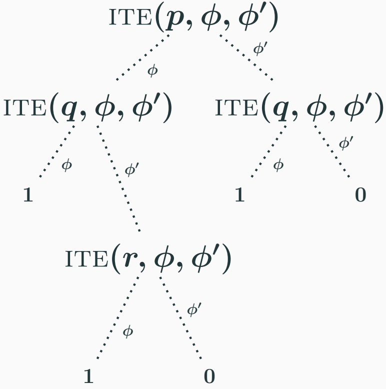
da expressão booleana $(\boldsymbol{p} \vee \boldsymbol{q}) \wedge (\boldsymbol{q} \vee \boldsymbol{r})$ é a mesma da sua arvore de decisão binária

Ilustração do argumento anterior

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



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



Estrutura de dados

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

- Uma tabela $\boldsymbol{T} : \boldsymbol{n} \mapsto \langle \boldsymbol{v}, \boldsymbol{t}, \boldsymbol{f} \rangle$
  - que associa a cada identificador $\boldsymbol{n}$ um nó com variável de teste $\boldsymbol{v}$, filho esquerdo $\boldsymbol{t}$ e filho direito $\boldsymbol{f}$
- Uma tabela inversa $\boldsymbol{T^{-1}} : \langle \boldsymbol{v}, \boldsymbol{t}, \boldsymbol{f} \rangle \mapsto \boldsymbol{n}$
  - que associa nós em identificadores
  - devido ao compartilhamento de sub-grafos
  - usada para garantir que os diagramas sejam reduzidos

Ilustração dessa estrutura de dados

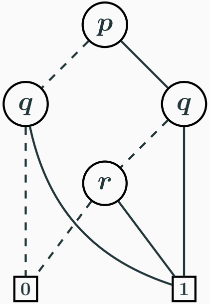


Tabela $T : n \mapsto \langle v, t, f \rangle$

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:

- $\boldsymbol{T(n) = T^{-1}(\langle v, t, f \rangle)} = \text{NULL}$ sempre que $\boldsymbol{(n, \langle v, t, f \rangle) \notin T}$
- A tabela $\boldsymbol{T}$ é uma variável global e $|\boldsymbol{T}|$ 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 $\mathbf{0}$ e $\mathbf{1}$
  - para garantir uniformidade, associa os nós terminais à uma variável auxiliar p_{m+1}

Pseudocódigo de INIT

```
procedure INIT(T, m)
    T ← {(0, ⟨m + 1, NULL, NULL⟩}),
        {(1, ⟨m + 1, NULL, NULL⟩})
```

Algoritmo de inserção de nós

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

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

Pseudocódigo de `INS`

```
function INS(T, v, t, f)
    if t = f then
        return t
    n ← T^{-1}(⟨v, t, f⟩)
    if n = NULL then
        n ← |T|
        T ← T ∪ {(n, ⟨v, t, f⟩)}
    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 $\boldsymbol{v}$ e os identificadores de seus filhos $\boldsymbol{t}$ e $\boldsymbol{f}$
- Se o nó $\boldsymbol{v}$ for redundante $(\boldsymbol{t} = \boldsymbol{f})$, devolve imediatamente o identificador do nó filho $(\boldsymbol{t})$
- Caso o nó $\boldsymbol{v}$ já tenha sido criado, devolve seu identificador
- Caso o nó $\boldsymbol{v}$ seja novo, cria-o e devolve o identificador

Pseudocódigo de BUILD

function $\mathrm{BUILD}(\textsc{ite}(v, \phi_t, \phi_f))$

    if $\phi_t, \phi_f \in \{0, 1\}$ **then**

        return $\mathrm{INS}(v, \phi_t, \phi_f)$

    if $\phi_f \in \{0, 1\}$ **then**

        return $\mathrm{INS}(v, \mathrm{BUILD}(\phi_t), \phi_f)$

    if $\phi_t \in \{0, 1\}$ **then**

        return $\mathrm{INS}(v, \phi_t, \mathrm{BUILD}(\phi_f))$

    return $\mathrm{INS}(v, \mathrm{BUILD}(\phi_t), \mathrm{BUILD}(\phi_f))$