A documentação da biblioteca PyEDA não é muito clara a respeito de como escolher a ordenação das variáveis para a construção de BDDs (no link http://pyeda.readthedocs.org/en/latest/bdd.html#variable-ordering). O que é mencionado lá é apenas o seguinte:

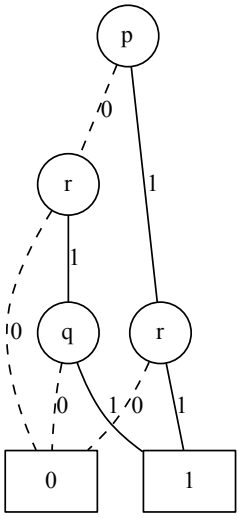
> PyEDA ordena todas as variáveis de forma implícita. Portanto não é possível criar um novo BDD reordenado suas entradas. Você pode, entretanto, renomear as variáveis utilizando o método *compose* para atingir o resultado desejado.
>
> (tradução livre do original em inglês: *PyEDA implicitly orders all variables. It is therefore not possible to create a new BDD by reordering its inputs. You can, however, rename the variables using the compose method to achieve the desired result*).

Por forma "implícita" entende-se que a biblioteca escolhe como a ordem a sequência em que as variáveis ocorrem em uma expressão booleana. Por exemplo, considere a expressão $\phi \equiv (p \wedge r) \vee (\neg p \wedge q \wedge r)$, se construida com o código a seguir:

```
from pyeda.inter import *
from subprocess import call
import os

########################################
# Função para exportação de diagramas em
# imagens (em diferentes formatos).
# Parâmetros:
#  b: ROBDD ou árvore sintática a ser
#  exportado.
#  fmt: Formato da exportação (entre os
#  formatos permitidos pela ferramenta
#  Graphviz (pdf, png, bmp, ps, etc).
#  file_name: Nome do arquivo a ser criado.
########################################
def Export2Image(b, fmt, file_name):
    # Exporta o diagrama para o Graphviz (linguagem Dot)
    with open('temp.gv', 'w') as hFile:
        hFile.write(b.to_dot())
    # Gera o PDF com o diagrama
    call(['dot', '-T' + fmt, 'temp.gv', '-o' + file_name])
    os.remove('temp.gv')

########################################
# Código principal
########################################
if __name__ == '__main__':

    ############################
    # Teste 1
    ############################

    # Cria a fórmula booleana
    f = expr("(p & r) | (~p & q & r)")
```

```
    # Cria o ROBDD da fórmula
    b = expr2bdd(f)

    # Gera a imagem do ROBDD em formato PDF
    Export2Image(b, 'pdf', 'bdd1.pdf')
```

Como as variáveis aparecem nessa expressão na ordem $[p, r, q]$, essa é a ordem utilizada na criação do BDD, de forma que o diagrama resultante desse código é o seguinte:

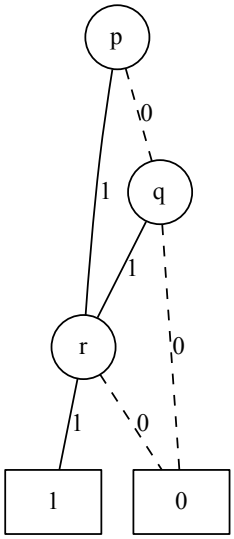


Caso se deseje utilizar a ordem $[p, q, r]$, pode-se inverter as cláusulas na expressão ou tentar utilizar o método `compose` como sugerido na documentação. Porém, há uma forma mais simples e direta de fazê-lo. Basta mapear as variáveis booleanas na ordem desejada *antes de construir a expressão booleana*. Por exemplo, considere a parte do código principal no programa anterior alterada da seguinte forma:

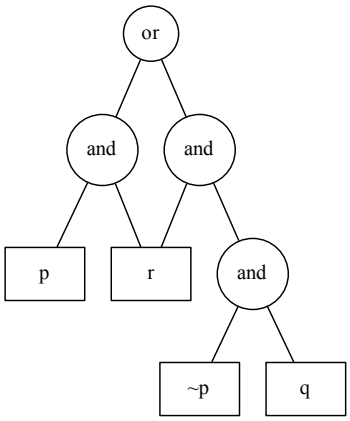
```
##########################################
# Código principal
##########################################
if __name__ == '__main__':

    ############################
    # Teste 2
    ############################

    # Cria as variáveis
    # (a ordem usada aqui é a que vai ser usada no ROBDD!)
    p, q, r = map(exprvar, 'pqr')
```

```
# Cria a fórmula booleana
f = (p & r) | (~p & q & r)

# Cria o ROBDD da fórmula
b = expr2bdd(f)

# Gera a imagem da árvore sintática da fórmula em formato PDF
Export2Image(f, 'pdf', 'arv-sintatica.pdf')

# Gera a imagem do ROBDD em formato PDF
Export2Image(b, 'pdf', 'bdd2.pdf')
```

Esse código cria variáveis ao invés de definir a fórmula por meio de uma string. Assim, quando a expressão é definida a biblioteca vai utilizar a ordem utilizada no parâmetro da chamada de `exprvar` (nesse exemplo: 'pqr'). Essa forma também permite reusar as variáveis em outras expressões (como variáveis em Python). Observe como o resultado desse novo programa é um diagrama mais reduzido do que o anterior, devido à ordem escolhida:



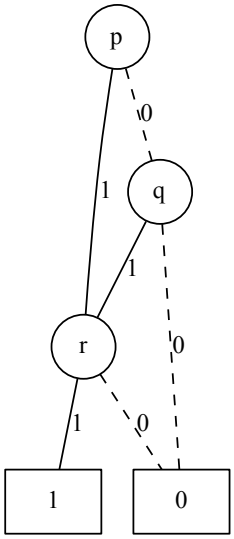
Observe também que, nos dois exemplos anteriores, há uma clara distinção entre o que é uma expressão booleana (uma fórmula) e um BDD. A função `expr2bdd` converte uma expressão em um BDD, mas é possível usar o mesmo comando `dot` para gerar também uma imagem da árvore sintática de uma expressão:



Finalmente, há também a opção de trabalhar diretamente com BDDs ao invés de criar expressões booleanas e convertê-las para BDDs. Para isso, ao invés de utilizar a função `exprvar` (ou `exprvars`, ao se criar uma matriz de variáveis), utiliza-se a função `bddvar` (ou `bddvars`, equivalente para se criar uma matriz). O programa a seguir é equivalente ao anterior, só que trabalha diretamente com BDDs:

```
##########################################
# Código principal
##########################################
if __name__ == '__main__':

    ############################
    # Teste 3
    ############################

    # Cria as variáveis para BDDs
    # (a ordem usada aqui é a que vai ser usada no ROBDD!)
    p, q, r = map(bddvar, 'pqr')

    # Cria a fórmula booleana diretamente em ROBDD
    b = (p & r) | (~p & q & r)

    # Gera a imagem do ROBDD em formato PDF
    Export2Image(b, 'pdf', 'bdd3.pdf')
```

E gera o mesmo ROBDD, devido à ordem escolhida $[p, q, r]$:


p
0
1
q
1
r
0
1
0
1
0