# Expected Shapley Value on deterministic-Decomposable (dD) circuits

November 4, 2023

# Contents

# 1 Node (class)

The **Node** class is used as a building block of the circuits.
It contains 6 attributes:

- **id_counter:** This is used as the unique identity of the node. It increases by 1 every time a new node is declared.
- **gate:** This holds the expression of the node in a tuple format.

  An expression is given as below:

  - A: 'A' (a variable)
  - $\neg$ A: ('NOT', 'A')
  - A $\wedge$ B: ('AND', 'A', 'B')
  - A $\vee$ B: ('OR', 'A', 'B')
  - A $\vee\neg$B: ('OR', 'A', ('NOT','B'))

- **node_type:** This contains the type of the node according to its operations. The types are:

  - **'OR':** A disjunction node.
  - **'AND':** A conjunction node.
  - **'NOT':** A negation node.
  - **'leaf'**: A leaf node.
  - **'constant':** A node that contains either **True** or **False** values.

- **children:** A list containing the children nodes of the node. **constant** nodes and **leaf** nodes contain no children. A **NOT** node contains only one child. **AND** and **OR** nodes contain two children each.
- **value: constant** nodes contain either **True** or **False** values. All other nodes contain value **None**.
- **evaluated:** Flag used for computation purpose. Contains Boolean values.

## 2 parse (function)

**parse** method is used to convert an expression into a circuit. The expressions are given as mentioned above in the **gate** in **Node** [1] class.
**Inputs:**

- **expression:** The expression in **smooth dD** form to parse.
- **exps:** A **dictionary** containing already parsed sub-expressions.
- **nodes:** A **list** containing already computed node ids.

**Output:** A circuit of **Node** class [1] parsed from the input expression.
To parse a new expression, the dictionary and the list should be empty.

## 3 input_gates (function)

**Inputs:**

- **node:** A **smooth dD node** or **smooth dD circuit** of class **Node**[1]

**Output:** A list of variables associated to the node/circuit.

## 4 delta (function)

**Inputs:**

- **node:** A **smooth dD node** or **smooth dD circuit** of class **Node**[1].
- **data: Dataset** in pandas dataframe format containing the variables and their probabilities having two columns named “t" and “p". Column “t" should contain the variable names and column “p" should contain the probabilities of the variables.
- **deltas: dictionary** that keeps track of the already computed nodes (memoization purpose).

To compute the $\delta$ of a node, the **deltas** dictionary should be empty.
**Output:** The $\delta_k^g$ values.

## 5 alpha (function)

**Inputs:** Takes similar inputs as **delta** [4] function.
**Output:** The $\alpha^g$ values.

# 6 conditioned_dD (function)

**Inputs:**

- **node:** A **smooth dD node** or **smooth dD circuit** of class **Node**[1].
- **fixed_var:** A list of variables (as strings) to condition on.
- **values:** The values (**True** or **False**) to assign to the variables.

**Output:** The conditioned node/circuit.

# 7 EShap (function)

**Inputs:**

- **node:** A **smooth dD node** or **smooth dD circuit** of class **Node**[1].
- **x:** The **variable name** as string.
- **data:** The **dataset** as mentioned in **delta** [4].

**Output:** The expected Shapley value of the variable in the input dataset.

# 8 Example usage

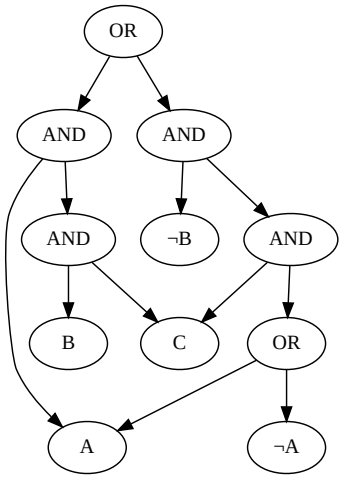


Figure 1: Example dD Circuit

```
from dDC_compute import input_gates, EShap, parse, compute,
    EShap_naive
import pandas as pd
import time

exp = ("OR", ("AND", "A", ("AND", "B", "C")), ("AND", ("NOT",
    "B"), ("AND", "C", ("OR", "A", ("NOT", "A")))))
query = parse(exp, {}, [])
data = pd.DataFrame({"t": ["A","B","C"], "p":[1,1,1]})
shap_sum = 0
for x in ['A','B','C']:
    print(x)
    t1 = time.time()
    eshap = EShap(query, x, data)
    t1 = time.time() - t1
    t2 = time.time()
    eshap_naive = EShap_naive(query, x, data)
    t2 = time.time() - t2
    print(eshap_naive, "Naive", "Time:", t2, "sec")
    print(eshap, "Ours", "Time:", t1, "sec", '\n')
    shap_sum += eshap
print('Shap Sum:',shap_sum)
print("All true query value:", float(compute(query,{"A":True,
    "B":True, "C":True}).value))
```

```
Output:
A
0.3333333333333333 Naive Time: 0.00803828239440918 sec
0.3333333333333333 Ours Time: 0.0053958892822265625 sec

B
-0.16666666666666666 Naive Time: 0.007795095443725586 sec
-0.16666666666666666 Ours Time: 0.003361940383911133 sec

C
0.8333333333333333 Naive Time: 0.008202075958251953 sec
0.8333333333333333 Ours Time: 0.005302906036376953 sec

Shap Sum: 0.9999999999999999
All true query value: 1.0
```