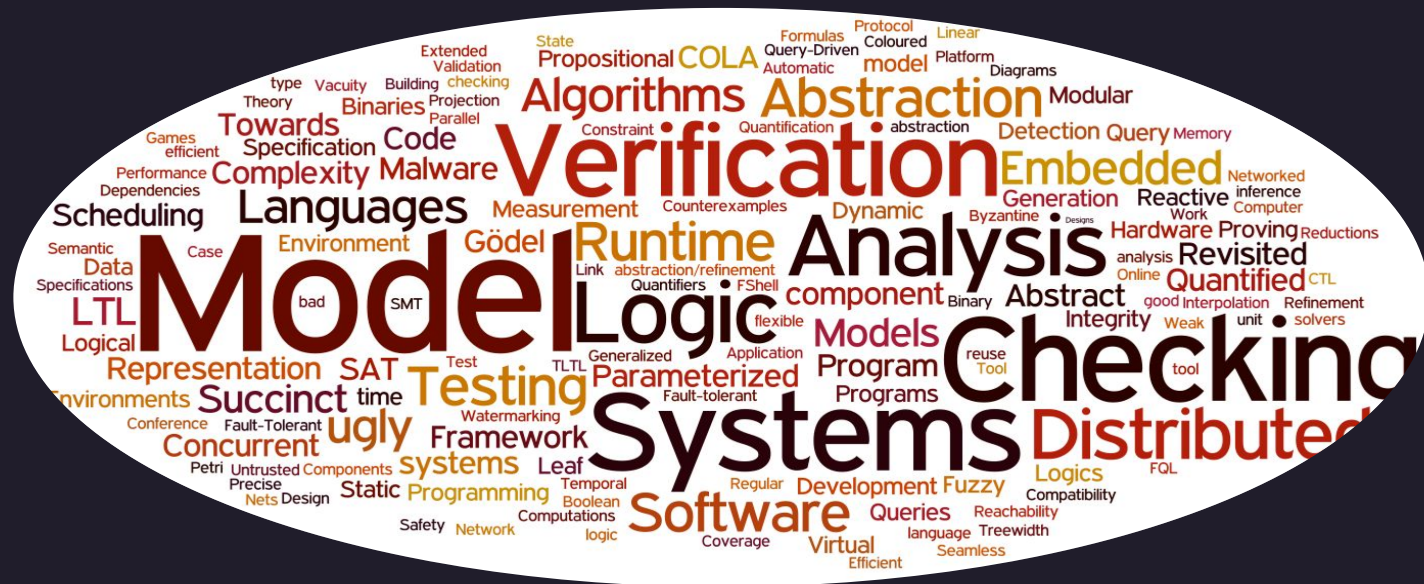
# Tutorial 2 : BDD HANDS-ON

**CS60030 Formal Systems**

PALLAB DASGUPTA,
FNAE, FASc,
A K Singh Distinguished Professor in AI,
Dept of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Email: pallab@cse.iitkgp.ac.in
Web: http://cse.iitkgp.ac.in/~pallab




INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

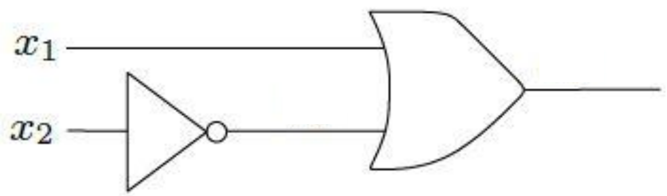
FMSAFE
FORMAL METHODS FOR SAFETY CRITICAL SYSTEMS

# Hands-on Session (pre-requisites)

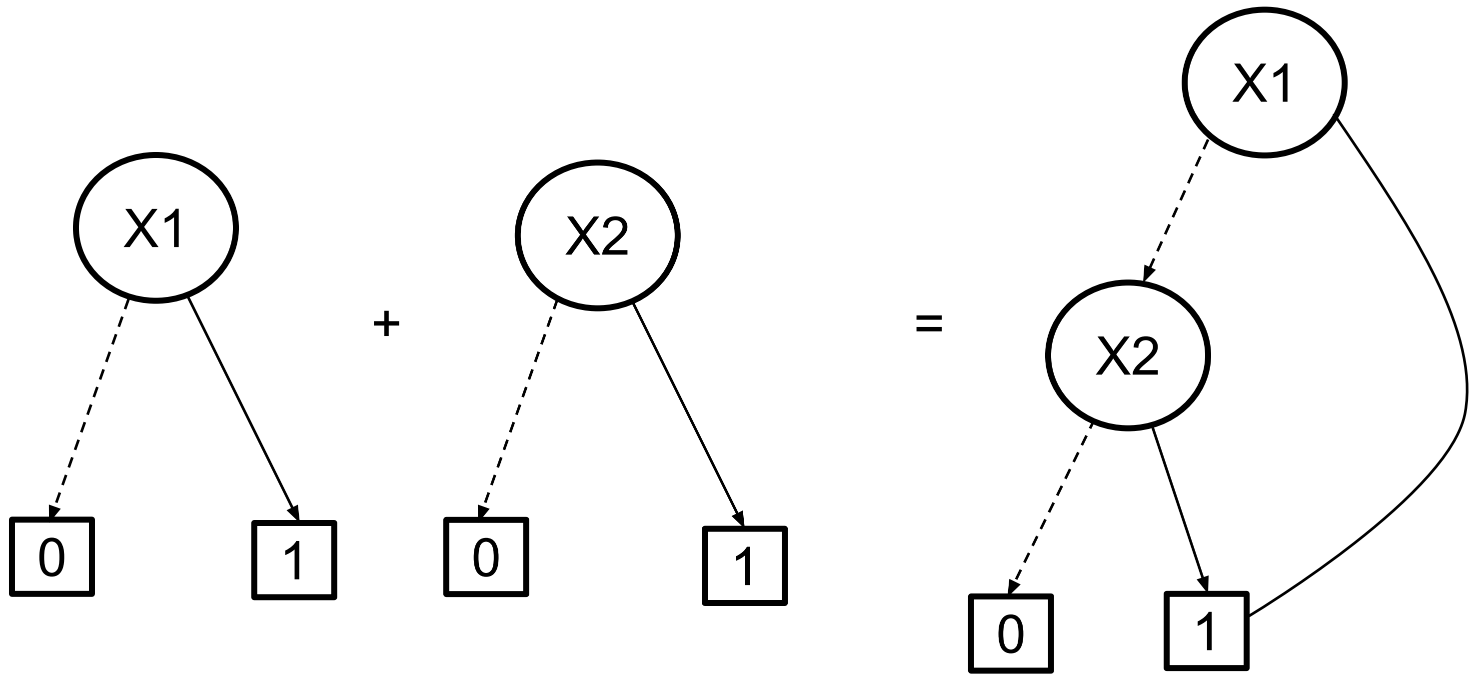
- **BDD solvers (CUDD) installed in the machines.**
  - Download CUDD version 3.0.0 from http://davidkebo.com/cudd and Untar.
  - Run the following
    - ./configure
    - make
    - make check
    - sudo make install
  - Include CUDD include libraries in C/C++ path
    - export CPATH=/<path-to-cudd>/:/<path-to-cudd>/cudd:/<path-to-cudd>/util/
  - To build add -lcudd -lutil -lm to the gcc command

# Gate Level Circuit to BDDs

- **Each input of the circuit is a BDD.**
- **Each gate becomes an operator that produces a new BDD.**
- **Example:**




$f = x_1 + \neg x_2$




BDD for f

# Using CUDD

- CUDD is a a C/C++ library for creating different types of decision diagrams (BDDs, ZDDs, ADDs).
- In order to use CUDD you must include two header files

```
#include "cudd.h"

#include "util.h"
```

- You should link libcudd.a, libmtr.a, libst.a, and libutil.a to your executable.

```
gcc -o main main.c -lcudd -lutil -lm
```

- To use the functions in the CUDD package, one has first to Initialize a DdManager using Cudd_Init()

```
DdManager *manager;

manager = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
```

- The constant 1 is returned by Cudd_ReadOne. The BDD logic 0 is obtained by complementation (Cudd_Not) of the constant 1

# Using CUDD

- **CUDD has a built-in garbage collection system. When a BDD is not used anymore, its memory can be reclaimed.**
- **To facilitate the garbage collector, we need to "reference" and "dereference" each node in our BDD:**

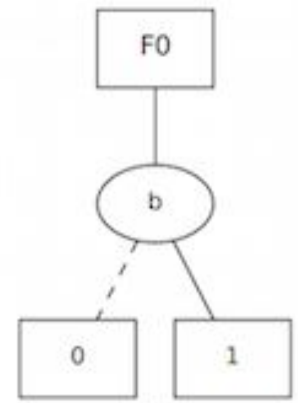
```
Cudd_Ref(DdNode*)
Cudd_RecursiveDeref(DdNode*)
```

- **The DdNode is the core building block of BDDs. New DdNodes can be created using the Cudd_bddNewVar function.**

```
DdNode *bdd = Cudd_bddNewVar(DdManager*)
```

- **Sample Program**

```
// This program creates a single BDD variable
int main (int argc, char *argv[])
{
    DdManager *gbm; /* Global BDD manager. */
    char filename[30];

    gbm = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
    DdNode *bdd = Cudd_bddNewVar(gbm); /*Create a new BDD variable*/
    Cudd_Ref(bdd); /*Increases the reference count of a node*/
    bdd = Cudd_BddToAdd(gbm, bdd); /*Convert BDD to ADD for display */

    sprintf(filename, "./bdd/graph.dot"); /*Write .dot filename*/
    write_dd(gbm, bdd, filename);  /*Write the dd to a file*/
    Cudd_Quit(gbm);
    return 0;
}
```



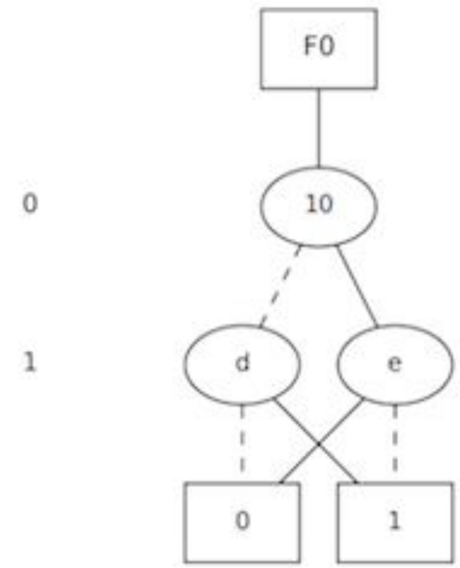
# BDD of Boolean functions

- **CUDD has inbuilt functions for expressing Boolean operations.**

```
Cudd_bddXor(DdManager*, DdNode*, DdNode*)

Cudd_bddAnd(DdManager*, DdNode*, DdNode*)

Cudd_bddOR(DdManager*, DdNode*, DdNode*)

Cudd_bddXnor(DdManager*, DdNode*, DdNode*)

Cudd_bddNand(DdManager*, DdNode*, DdNode*)

Cudd_bddNor(DdManager*, DdNode*, DdNode*)

Cudd_bddNot(DdNode*)
```

# Implementing XOR Using CUDD

```
int main (int argc, char *argv[])

{
 char filename[30];
 DdManager *gbm; /* Global BDD manager. */
 gbm = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0);
 DdNode *bdd, *x1, *x2;
 x1 = Cudd_bddNewVar(gbm); /*Create a new BDD variable x1*/
 x2 = Cudd_bddNewVar(gbm); /*Create a new BDD variable x2*/
 bdd = Cudd_bddXor(gbm, x1, x2); /*Perform XOR*/
 Cudd_Ref(bdd);          /*Update the reference count*/
 bdd = Cudd_BddToAdd(gbm, bdd);
     sprintf(filename, "./bdd/graph.dot"); /*Write .dot filename*/
     write_dd(gbm, bdd, filename);
     Cudd_Quit(gbm);
     return 0;
}
```



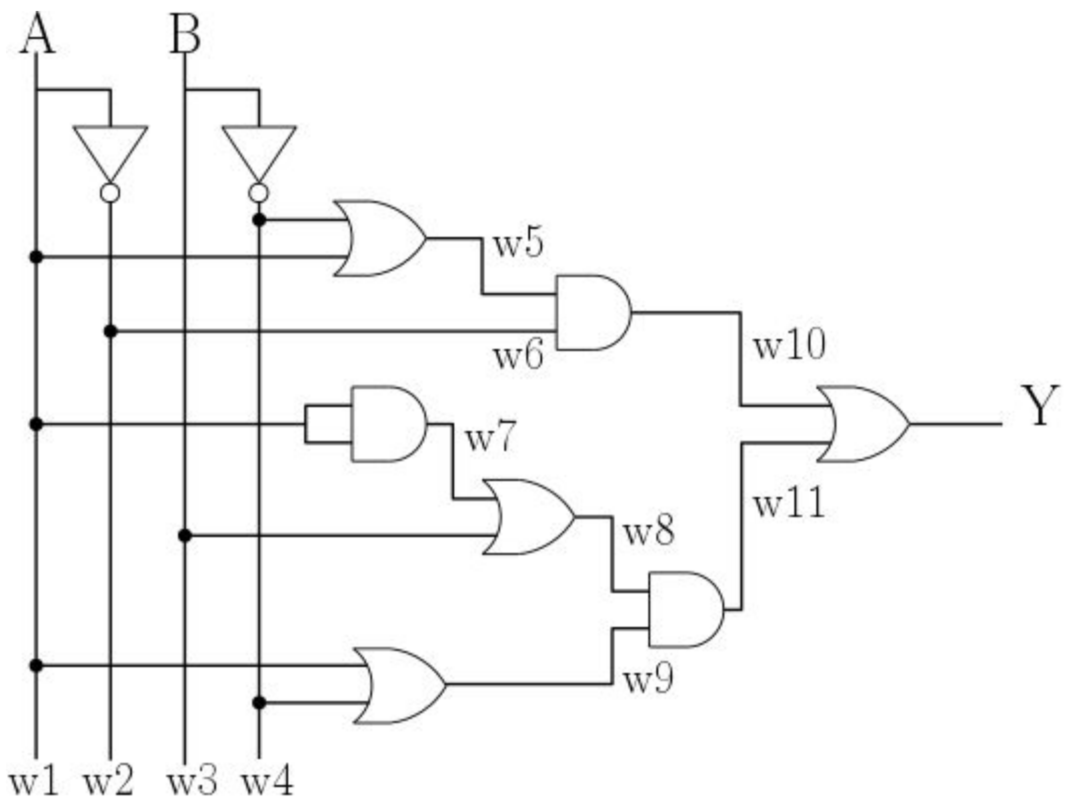
# Comparing Logic Implementations

- Are two Boolean logics F and G the same?
  - Build BDD for F.
  - Build BDD for G.
  - Compare pointers to roots of F, G.
  - If pointers are same, F == G.
- What inputs make functions F, G give different answers?
  - Build BDD for F.
  - Build BDD for G.
  - Build the BDD for H = F xor G.
  - Check if H is satisfiable or not.

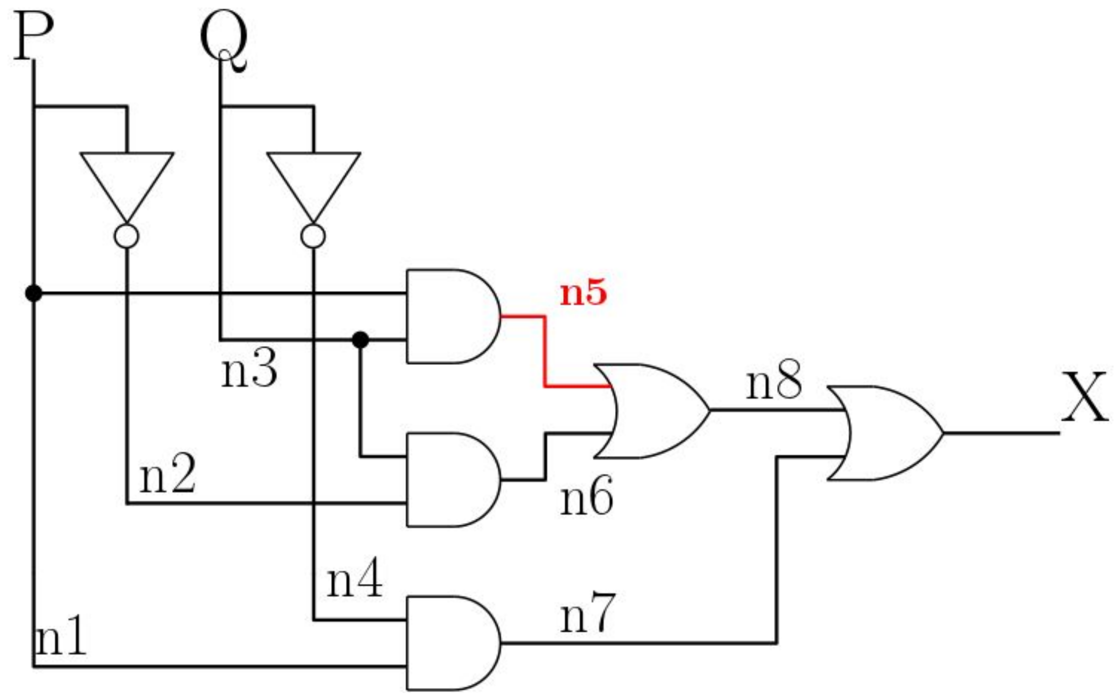
# Tautology Checking and Satisfiability with BDDs

- **Tautology Checking**
  - **The function will be reduced to one node pointing to 1.**
- **Satisfiability Checking**
  - **Any path from root to “1” leaf is solution!**

# Observe the two circuits




**Circuit-1**




**Circuit-2**

**Inputs A and B are equivalent to inputs P and Q.**

# Assignments on BDD

1. Represent Circuit-1 and Circuit-2 in as BDDs.

2. Check whether the two circuits are equivalent or not.

3. Find the input condition(s) for which the output is 1.