A Minimalistic BDD Library

▶ For teaching/learning purposes

▶ Designed for ease of use
   (there are more efficient libraries)

▶ only 556 lines of C++
   (compare to cudd, which has 117k lines)
Data Structures

▶ A class for nodes
  ▶ With pointers to the two children
  ▶ With a reference counter

▶ The nodes are stored in a list of nodes in a BDD manager class

▶ The manager also contains:
  ▶ A list of the variables (with a label)
  ▶ The hash table for the nodes
The BDD Node Class

```cpp
class BDDnode {
    class miniBDD_mgr *mgr;
    unsigned var, node_number, reference_counter;
    BDD low, high;

    inline void add_reference();
    void remove_reference();
};
```

There is also a (trivial) constructor.
The BDD Manager Class

```cpp
class miniBDD_mgr {
  public:
    BDD Var(const std::string &label);

    inline const BDD &True();
    inline const BDD &False();

  protected:
    typedef std::list<BDDnode> nodest;
    nodest nodes;

    struct var_table_entryt { std::string label; };
    typedef std::vector<var_table_entryt> var_tablet;
    var_tablet var_table;

  ...
```

There is also a constructor (which sets up True/False), and some methods to dump the node table.
The BDD Manager Class (Part II)

```cpp
1 class miniBDD_mgr {
2   ...
3
4   // this is our reverse-map for nodes
5   struct reverse_keyt {
6       unsigned var, low, high;
7   };
8
9   std::map<reverse_keyt, BDDnode *> reverse_map;
10
11   // create a node (consulting the reverse-map)
12   BDD mk(unsigned var,
13      const BDD &low, const BDD &high);
14 }
```
The Interface (Part I)

```cpp
class BDD {
public:
  // Boolean operators on BDDs
  BDD operator !( ) const;
  BDD operator ^(const BDD &) const;

  // copy operator
  inline BDD &operator=(const BDD &);

protected:
  class BDDnode *node;
};
```

There are more Boolean operators (&, |, ==). This is essentially only one pointer, so copying is inexpensive.
There are also some methods to obtain information about a BDD:

```cpp
class BDD {
public:

  ...  
  inline bool is_constant() const;
  inline bool is_true() const;
  inline bool is_false() const;

  inline unsigned var() const;
  inline const BDD &low() const;
  inline const BDD &high() const;
  inline unsigned node_number() const;

  ...  
};
```
Using the Interface

```
#include "miniBDD.h"

int main() {
    miniBDD_mgr mgr;

    BDD final =
        mgr.Var("x") & mgr.Var("y");
```

This produces:

```
y
x
```

**Warning:** The `mgr.Var(...)` method doesn’t hash, so calling `mgr.Var("x")` twice will produce two different variables, both labelled "x".
Using the Interface

You can look at the BDDs or the node table with:

1. `void DumpDot(const std::ostream &out);`
2. `void DumpTikZ(const std::ostream &out);`
3. `void DumpTable(const std::ostream &out);`

This produces:

```
0 1
y      x
4      3

# | var | low | high
---|--|--|--
0 | 3 |
1 | 3 |
2 |   |   |   
3 | 2x | 0 | 1
4 | 1y | 0 | 3
```
The Implementation of mk

```c++
BDD miniBDD_mgr::mk(unsigned var, BDD low, BDD high) {
    if (low.node_number() == high.node_number())
        return low;

    reverse_keyt reverse_key(var, low, high);
    reverse_map::const_iterator it = reverse_map.find(reverse_key);

    if (it != reverse_map.end()) return BDD(it->second);

    unsigned new_number = nodes.back().node_number + 1;
    nodes.push_back(
        BDDnode(this, var, new_number, low, high));
    reverse_map[reverse_key] = &nodes.back();
    return BDD(&nodes.back());
}
```
The Implementation of apply

```c
BDD apply(bool (*fkt)(bool x, bool y),
           BDD x, BDD y)
{
    miniBDD_mgr *mgr=x.node->mgr;

    BDD u;

    if(x.is_constant() && y.is_constant())
        u=BDD(fkt(x.is_true(), y.is_true())?
                mgr->true_bdd:mgr->false_bdd);
    else if(x.var()==y.var())
        u=mgr->mk(x.var(),
                   apply(fkt, x.low(), y.low()),
                   apply(fkt, x.high(), y.high()));
    ...
    return u;
}
```