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


```
1  class BDDnode {
2    class miniBDD_mgr *mgr;
3    unsigned var, node_number, reference_counter;
4    BDD low, high;
5
6    inline void add_reference();
7    void remove_reference();
8  };
```

There is also a (trivial) constructor.

The BDD Manager Class


```
1  class miniBDD_mgr {
2  public:
3    BDD Var(const std::string &label);
4
5    inline const BDD &True();
6    inline const BDD &False();
7
8  protected:
9    typedef std::list<BDDnode> nodest;
10   nodest nodes;
11
12   struct var_table_entryt { std::string label; };
13   typedef std::vector<var_table_entryt> var_tablet;
14   var_tablet var_table;
15   ...
```

There is also a constructor (which sets up True/False), and some methods to dump the node table.

The BDD Manager Class (Part II)


```
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)


```
1  class BDD {
2  public:
3    // Boolean operators on BDDs
4    BDD operator !() const;
5    BDD operator ^(const BDD &) const;
6
7    // copy operator
8    inline BDD &operator=(const BDD &);
9
10 protected:
11   class BDDnode *node;
12 };
```

There are more Boolean operators (&, |, ==).
This is essentially only one pointer, so copying is inexpensive.

The Interface (Part II)

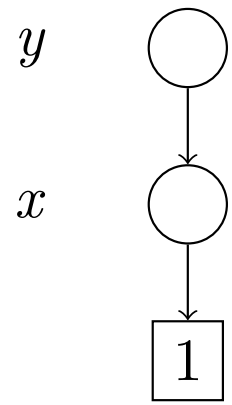
There are also some methods to obtain information about a BDD:

```
1  class BDD {
2  public:
3     ...
4     inline bool is_constant() const;
5     inline bool is_true() const;
6     inline bool is_false() const;
7
8     inline unsigned var() const;
9     inline const BDD &low() const;
10    inline const BDD &high() const;
11    inline unsigned node_number() const;
12    ...
13 };
```

Using the Interface

```
1  #include "miniBDD.h"
2
3  int main() {
4     miniBDD_mgr mgr;
5
6     BDD final=
7        mgr.Var("x") & mgr.Var("y");
```

This produces:



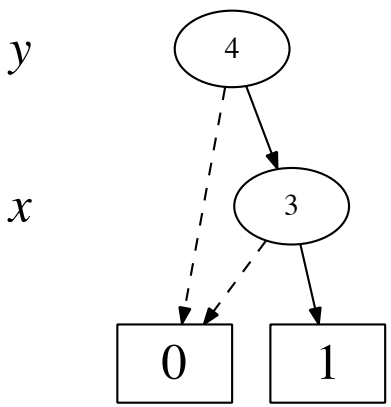
Warning: The $\mathrm{mgr.Var}(...)$ method doesn't hash,
so calling $\mathrm{mgr.Var}(\text{"x"})$ twice will produce two different variables,
both labelled "x".

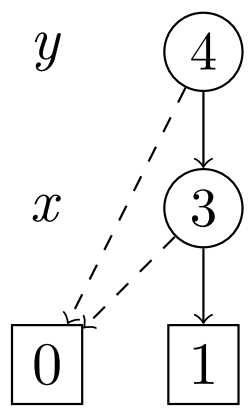
Using the Interface

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

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

This produces:





#	*var*	*low*	*high*
0	3		
1	3		
2	–	–	–
3	$2\,x$	0	1
4	$1\,y$	0	3

The Implementation of mk


```
1  BDD miniBDD_mgr::mk(unsigned var, BDD low, BDD high) {
2    if(low.node_number()==high.node_number())
3      return low;
4
5    reverse_keyt reverse_key(var, low, high);
6    reverse_mapt::const_iterator it=
7      reverse_map.find(reverse_key);
8
9    if(it!=reverse_map.end()) return BDD(it->second);
10
11   unsigned new_number=nodes.back().node_number+1;
12   nodes.push_back(
13     BDDnode(this, var, new_number, low, high));
14   reverse_map[reverse_key]=&nodes.back();
15   return BDD(&nodes.back());
16 }
```

The Implementation of apply

```
1  BDD apply(bool (*fkt)(bool x, bool y),
2            BDD x, BDD y)
3  {
4    miniBDD_mgr *mgr=x.node->mgr;
5
6    BDD u;
7
8    if(x.is_constant() && y.is_constant())
9      u=BDD(fkt(x.is_true(), y.is_true())?
10           mgr->true_bdd:mgr->false_bdd);
11   else if(x.var()==y.var())
12     u=mgr->mk(x.var(),
13               apply(fkt, x.low(), y.low()),
14               apply(fkt, x.high(), y.high()));
15   ...
16   return u;
17 }
```