

ECE 510 OCE BDDs and Their Applications

Lecture 6. Programming with BDDs

April 13, 2000

Alan Mishchenko

Overview

- Downloading and compiling the BDD package
- Starting the package and defining variables
- Building BDDs from elementary BDD representing variables in positive and negative polarity
- Most often used operations on BDDs
- Typical structure of an efficient BDD-based procedure
- Examples:
 - Counting the number of true assignments of a function
 - Getting one true assignment of a function
- The importance of hashing
- Homework problem

Downloading and Compiling

- <http://www.itu.dk/research/buddy/index.html>
- Read introductory pages of the manual
- If you are using Windows, fix the bug:
 - the header of the function in "kernel.c"
`int bdd_makenode(int level, int low, int high)`
 - should be
`int bdd_makenode(unsigned int level, int low, int high)`
- Change "kernel.h" accordingly. This is it.
- Compile the package as a static library and link it to your C/C++ project

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

3

Starting Package, Defining Variables

- `bdd_init(int <unique table size>, int <cache size>);`
- `bdd_setvarnum(int <number of variables>);`
- Typical values:
 - `<unique table size> = 10000`
 - `<cache size> = 1000`
 - `<number of variables> = 100`
- At the end of the program, do not forget to call `bdd_done ();`

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

4

Building BDDs

- BDDs for functions are created using calls to functions that return BDDs for elementary variables
- `bdd bdd_ithvar(int <variable number>);`
- `bdd bdd_nithvar(int <variable number>);`
- Representations of complex functions are built using the following operations: `NOT(!)`, `AND(&)`, `OR (|)`, `EXOR (^)`, `implication(>>)`, `sharp(-)`, etc. and function `bdd bdd_ite(bdd F, bdd G, bdd H);`
- For example, $F = x_0 \& x_1' + x_2$ is created like this

```
bdd F = bdd_ithvar( 0 ) & bdd_nithvar( 1 ) |  
        bdd_ithvar( 2 );
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

5

Getting Top-Variable, and Cofactoring

- To get the variable that is labeling the topmost node of the BDD, call function `int bdd_var(bdd F);`
- To get the cofactors w.r.t. the topmost variable, call functions `bdd bdd_low(bdd F);` `bdd bdd_high(bdd F);`
- To get a cofactor w.r.t. any variable or cube, call the function `bdd bdd_restrict(bdd F, bdd Cube);`

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

6

Structure of a BDD-based Procedure

```
<value_type> Function( bdd F )
{ // consider terminal cases
  if ( F == bddfalses ) return <simple_solution1>;
  if ( F == bddtrue ) return <simple_solution2>;
  // check cache for results
  // solve the sub-problems
  <value_type> F0 = Function( bdd_low( F ) );
  <value_type> F1 = Function( bdd_high( F ) );
  // derive the solution from the solutions of sub-problems
  <value_type> Result = Solve( F0, F1 );
  // insert the result into cache
  return Result;
}
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

7

Case Study: Counting Paths

```
int CountPaths( const bdd F )
{
  if ( F == bddfalses ) return 0;
  if ( F == bddtrue ) return 1;
  // check cache for results
  int R0 = CountPaths( bdd_low(F) );
  int R1 = CountPaths( bdd_high(F) );
  int Res = R0 + R1;
  // insert into cache
  return Res;
}
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

8

Case Study: Finding One True Path

```
bdd GetOnePath( const bdd F )
{
    if ( F == bddfalses ) return bddfalses;
    if ( F == bddtrue ) return bddtrue;
    if ( bdd_low(F) != bddfalses )
        // the low path is not zero, we follow it
        return bdd_nithvar( bdd_var(F) ) & GetOnePath( bdd_low(F) );
    else // if ( bdd_high(F) != bddfalses )
        // the low path is zero, we can only follow the high path
        return bdd_ithvar( bdd_var(F) ) & GetOnePath( bdd_high(F) );
}
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

9

Easy Implementation of Cache

```
#define TABLESIZE 11111
typedef struct HashEntry_tag {
    bdd F;
    int Res;
} HashEntry;
HashEntry HTable[TABLESIZE];
int HashSuccess = 0;
int HashFailure = 0;
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

10

Using Cache in Path Counting Function

```
int CountPaths( const bdd F )
{
    if ( F == bddfalses ) return 0;
    if ( F == bddtrue ) return 1;

    // check cache for results
    int HashValue = F.id() % TABLESIZE;
    if ( HTable[HashValue].F == F ) {
        HashSuccess++;
        return HTable[HashValue].Res;
    }
    HashFailure++;
}
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

11

Using Cache (continued)

```
int R0 = CountPaths( bdd_low(F) );
int R1 = CountPaths( bdd_high(F) );
int Res = R0 + R1;

// insert into cache
HTable[HashValue].F = F;
HTable[HashValue].Res = Res;
return Res;
}
```

April 13, 2000

ECE 510 OCE: BDDs and Their Applications

12

Homework Problem

- Practice building BDDs programmably from bdds representing elementary variables
 - for example, $F = x_0 \& x_1 + x_2 \& x_3 + x_4 \& x_5$
- Use the BDD package function `bdd_nodecount()` to determine the number of nodes in the function's BDD
- Write BDD-based procedure `ComputeProbability()` that takes one argument, a BDD for the given function, and return a floating point probability that the value of this function is 1 (this probability is the ratio of the number of true minterms over the number of all minterms)
- Test your procedure with and without cache for functions $F = \text{OR}(x_{2i} \& x_{2i+1})$, $0 \leq i < n$, $n = 20, 40$, etc.