BDDTCL: An Environment for Visualizing and Manipulating Binary Decision Diagrams

Kurt E. Partridge

University of Washington
Seattle, WA 98195 USA
+1 206 685 4087
kepart@cs.washington.edu

ABSTRACT

A Binary Decision Diagram (BDD) is a data structure used in hardware verification to represent boolean expressions. Most BDD implementations provide only textual output and require the user to interact with them using a compiled programming language. BDDTCL provides an interpreted language for manipulating BDDs and a graphical viewer for manipulating and visualizing them. BDDTCL can draw BDDs with over 4,400 nodes; much larger than can be easily drawn by hand. Two users, a hardware design researcher and a student unfamiliar with BDDs, provided feedback about BDDTCL's effectiveness for understanding systems modeled by BDDs. These users also provided feedback for improving BDDTCL.

Keywords: data structure visualization, binary decision diagrams, hardware verification tools

INTRODUCTION

A Binary Decision Diagram (BDD) is a directed acyclic graph (DAG) for representing a boolean expression [1]. Each node in the graph represents a variable, and has two outgoing arcs: one taken if the variable is true, the other if it is false. The order of the variables on every path through the DAG must be the same. BDDs are useful because many boolean operations (such as testing for equivalence or evaluating an expression given a variable assignment) require execution time that is only a polynomial function of the number of variables in the expression. Other boolean expression representations, such as truth tables and clauses, require execution time exponential in the number of variables. BDDs have proven valuable for practical hardware circuit verification because they can represent large state spaces efficiently (in some cases as many as 10^{20} states!).

However, BDDs have drawbacks. They are very sensitive to the order of the variables. A poor variable ordering can greatly increase a BDD's size and raise execution times up to exponential levels. Also, it takes a while to learn how to use BDDs and it is hard to develop an intuition for good variable orderings.

```
zfi:   1 #
aaci:  1 #
zai:   1 #
s_1_0: 2 ##
s_1_1: 4 ####
s:     4 ####
s_3_0: 2 ##
s_3_6: 2 ##
fi:    2 ##
s_3_1: 2 ##
s_3_2: 2 ##
s_3_4: 2 ##
s_4_0: 2 ##
xi:    4 ####
s_5_0: 2 ##
yi:    4 ####
s_1_2: 2 ##
s_7_0: 2 ##
zi:    4 ####
...
Total: 75
```

```
if po
  0
else if !po
  if s_2_1
    0
  else if !s_2_1
    if s_3_5
      if zvo
        0
      else if !zvo
        5: if aci
          if vi
            !t
          else if !vi
            0
          endif vi
        else if !aci
          0
        endif aci
        ...
```

Figure 1: *Sample node profile (left column) and if-then-else representation (right two columns). The node profile produces a histogram of the number of nodes at each level. The if-then-else format completely represents the BDD.*

BDDTCL

BDDTCL is a graphical interface for viewing and manipulating BDDs. Existing BDD implementations provide only textual descriptions of a BDD's structure. For example, bddlib [4], provides two output formats: the node profile and the if-then-else form (see Figure 1). While the node profile is helpful for displaying a BDD's size and the effectiveness of a variable ordering, it is only an abstract description of the BDD, not the complete structure. The if-then-else format completely describes the BDD, but it is also lengthy and makes comparing two BDDs difficult when changes are buried deep within the output.

BDDTCL draws BDDs in a graphical format similar to the format used in research papers (see Figure 2). BDDTCL uses bddlib to manage the BDDs and Dot [3], a directed-graph layout program, to compute the geometry.

BDDTCL provides several features for visualizing large BDDs. First, BDDTCL supports traditional zooming. Since fonts become hard to read at small magnifications, the name of the node under the mouse pointer is displayed in a separate box. Second, instead of the research literature's solid or dashed arcs, BDDTCL colors them red or black. We found this much easier to read, especially at small resolutions. Finally, for

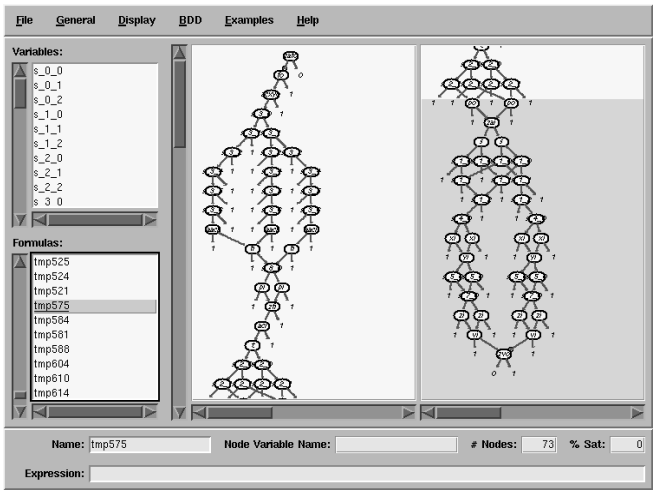

Figure 2: *A screen snapshot of BDDTCL.*

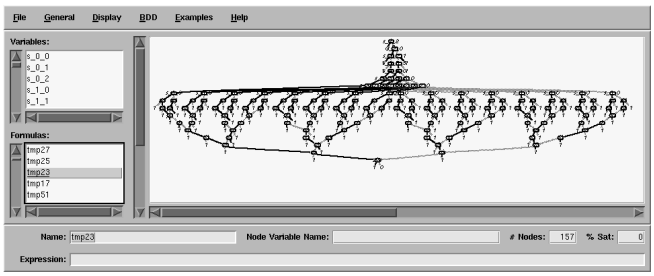

Figure 3: *Another screen snapshot of BDDTCL. This BDD has 157 nodes. Its if-then-else representation would require 617 lines (over nine pages) to print.*

BDDs that are thin and tall, BDDTCL supports a multicolumn display. To help give the user context between columns, different shadings are used to indicate overlapping areas. BDDTCL has drawn BDDs with over 4,400 nodes.

The user can create a BDD in many ways. The most direct is to type a boolean equation (such as $((grad_student \,\&\&\, free_lunch) \,||\, !grad_student)$) into an entry box at the bottom of the window. Larger equations may be typed into a command-line prompt or read in from a file. BDDTCL can also display other programs' BDDs if given the address of a BDD's root node.

BDDTCL also supports manipulations of BDDs. Bddlib and other BDD implementations with C interfaces require users to recompile if they make a change. This discourages iterative development because the user must wait during every development cycle for the compilation to finish. BDDTCL, on the other hand, is based on Tcl, which is interpreted, so recompilation is not necessary and iterative development is encouraged. For example, BDDTCL allows users to repetitively experiment with the parameters of bddlib's heuristic variable reordering algorithms so they can be fine-tuned.

All of bddlib's features may be accessed through Tcl commands, which may be read either from the command line or a file. Examples of such commands include combining and comparing BDDs by boolean operations (and, or, etc.), finding a satisfying assignment of variables to a BDD, printing BDD statistics or node profiles, deleting BDDs, and reordering variables. The graphical interface also can invoke many of these commands.

CASE STUDIES

BDDTCL was tested with two users: one who had been using bddlib for several months, and another who was still learning how BDDs worked. The expert used BDDTCL to display BDDs that were a part of an asynchronous circuit design. Prior to using BDDTCL, he had been using only textual output. He found that BDDTCL helped give him a more intuitive understanding of what parts of his application BDDs could represent efficiently and what parts they could not.

He especially liked the interface features for handling large graphs: node label identification by the mouse, the color-coded arcs, zooming, and the multicolumn display. He also made several suggestions for simplifying the graph, such as replicating the terminal nodes to avoid a large set of arcs pointing to a single node at the bottom of the graph.

The novice user found that BDDTCL solidified his understanding of the equivalence between a boolean expression and its BDD. He also said that the visual feedback made it much easier for him to understand the effects of various operations, and that the graphical user interface and interpreted command language made it easy to explore these operations.

FUTURE WORK

We plan to continue to improve BDDTCL by getting feedback from additional users, both those already familiar with BDDs and those still learning about them. We would also like to study additional large graph management techniques such as a fisheye view [2].

ACKNOWLEDGMENTS

Thanks go to Francesmary Modugno for her considerable support and suggestions on the paper and tool; to Steve Burns for helpful feedback, providing the case study, and being the user; to Jon Reese and James Landay for paper feedback and to Sean Sandys and Brendan Mumey for additional suggestions about BDDTCL; to David Long for writing bddlib; and to my advisor Nancy Leveson for her support and encouragement.

REFERENCES

1. R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

2. George W. Furnas. Generalized fisheye views. In *In proceedings of CHI, 1986*, pages 16–23, New York, 1986. ACM.

3. Eleftherios Koutsofios and Stephen C. North. Drawing graphs with *dot*. *http://www.research.att.com:80/orgs/ssr/book/reuse/*, 1993.

4. David E. Long. bddlib—a binary decision diagram (bdd) package. *http://www.cs.cmu.edu/afs/cs/project/modck/pub/www/bdd.html*, 1993.