Ben-Gurion University of the Negev

Communication Systems Engineering Department

381-1-0112 Functional Programming in Concurrent and Distributed Systems

Sequential Erlang - Assignment

Introduction

In this assignment, you are asked to implement an automatic construction machine of a Binary Decision Diagram (BDD) to represent a Boolean function, so a user is able to get a BDD representation of a function within a single call to your machine.

BDD is a tree data structure that represents a Boolean function. The search for a Boolean result of an assignment of a Boolean function is performed in stages, one stage for every Boolean variable, where the next step of every stage depends on the value of the Boolean variable that's represented by this stage.

A BDD tree is called *reduced* if the following two rules have been applied to it:

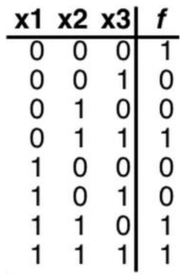
1. Merge any isomorphic (identical) sub-graphs
2. Eliminate any node whose two children are isomorphic

The construction of BDD is based on Shannon expansion theory:

$f(x_1, x_2, \dots, x_n) = x_1 \cdot f(1, x_2, x_3, \dots, x_n) + \overline{x_1} \cdot f(0, x_2, x_3, \dots, x_n)$

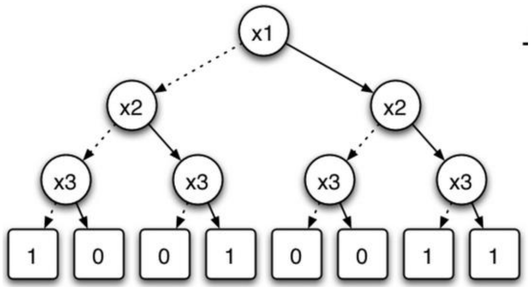
Example

The Boolean function $f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\,x_2 + x_2\,x_3$ with the following truth table:

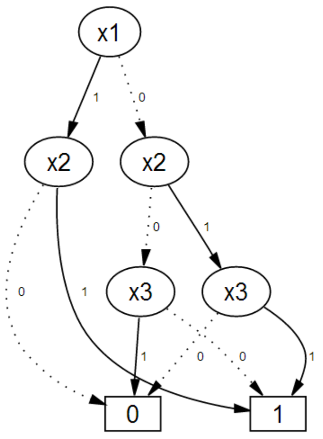


x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The BDD tree representation for this Boolean function:



By applying the reduction rules, we get:



Note that the results are the same for both BDD trees, for every assignment of the Boolean function.

Technical Details

Your mission is to implement two functions:

1. **spec exp_to_bdd**(BoolFunc, Ordering) → BddTree.
    - *a.* The function receives a Boolean function and returns the corresponding BDD tree representation for that Boolean function.
    - b. The returned tree must be the one that is the most efficient in the manner specified in variable **Ordering**.
        - i. Variable **Ordering** can be one of the following atoms: **tree_height**, **num_of_nodes** or **num_of_leafs**.
        - ii. In order to extract the most efficient tree you should **Compare all the possible permutations** of BDD trees.
        - iii. *Permutations:* let's assume that you are asked to convert a Boolean function that has 3 Boolean arguments: $f_s(x_1, x_2, x_3)$. f_s could be expanded in 3! different ways which means 6 BDDs. each BDD is a result of Shannon expansion applied to variables in distinct order. For f_s all the possible permutations are: $\{\{x_1, x_2, x_3\}, \{x_1, x_3, x_2\}, \{x_2, x_1, x_3\}, \{x_2, x_3, x_1\}, \{x_3, x_2, x_1\}, \{x_3, x_1, x_2\}\}$
    - c. The returned tree must be reduced by Rule 1. Read more about in BDD Introduction.
2. **spec solve_bdd**(BddTree, [{x1,Val1},{x2,Val2},{x3,Val3},{x4,Val4}]) → Res.

    The function receives a BDD tree and a list of values for every Boolean variable that's used in the Boolean function and returns the result of that function, according to the given BDD tree.

    Given values may be either in the form of **true/false** or **0/1**.

    The list of variables' values (the second argument of solve_bdd function) could be given at any order. Therefore, the function should be capable to handle any given order.

When returning from each function call, **the execution time needs to be printed**.

Note: signatures of these functions have to be exactly the same as those specs. This assignment will be checked by test program. Any mismatches might lead to points lose.

Input definition:

A Boolean function is given using the following format:

- Each operator will be described by one of the following tuples:
  - {'not', Arg}
  - {'or', {Arg1, Arg2}}
  - {'and', {Arg1, Arg2}}
- No more than two arguments will be evaluated by a single operator
- Example: the Boolean function

$$f(x_1, x_2, x_3) = x_1\,\overline{x_2} + x_2\,x_3 + x_3$$

  Will be represented as

  {'or',{ {'or',{ {'and',{ x1 , {'not', x2} }} , {'and',{ x2 , x3 }} }} , x3 }}

Your BDD tree needs to follow the following structure:

- An empty tree (root only)

  {Val}
- A node with both sons exist

  {Left, Right}
- A node with a single son exists (for example, the left one)

  {Left, {Val}}
- A leaf (represents a result)

  {Val}

  You can pick any data structure to be the tree's data structure but you must suppoly detailed explanation in your report why it was picked and also count advantages and disadvantages.

<u>Important notes</u>

1. Make sure that your code is well documented, efficient and tested.
2. The machine needs to identify the variables that take part in the Boolean expression, there is no guarantee that the variables will follow the structure of **x1,x2,…** or any other structure.
3. You are required to submit a short design report to your project with detailed explanation about your BDD tree structure.
4. You are required to add a project report.
   a. The report is performed on the following Boolean function:
   $$f_r(x_1,x_2,x_3,x_4) = x_1\,\overline{x_2}\,x_3 + \overline{x_1\overline{x_3}(\overline{x_4}+x_2)} + \overline{x_4x_1}$$
   b. Measure the Time elapsed for your program to convert f_r Boolean function to a BDD. Repeat on this stage each time with different ordering argument.
   Write the results in table.
   c. To each of the trees created in 'b' apply solve_bdd function with all the possible inputs. Measure the time it took to each tree. Write the results in table and mention which is the most efficient.
   d. Explain the results, You can use any graphical tool to generate graphs and plots which support your explanations.
   e. Write your conclusions from working on this assignment.
   f. Suggest an approach of a parallel computing of this problem (no implementation, only explanation).

<u>Submission Procedure</u>

Submit a single .zip: exf_<ID>.zip.

Main module is named as: exf_<ID>.erl

Wrong submission procedure leads <u>to penalty of 10 points!</u> which means maximal possible grade is 90.