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. The API should allow the user getting a BDD representation of a function within a single call to your machine and select the data structure it uses.

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 (bonus)
2. Eliminate any node whose two children are isomorphic

The construction of BDD is based on Shannon expansion theory:

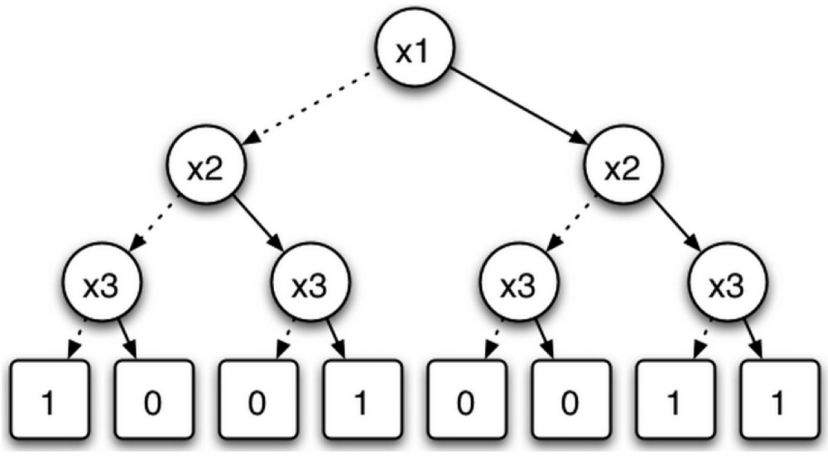
$f(x_1, x_2, \dots, x_n) = x_1 \cdot f(1, x_2, x_3, \dots, x_n) + \overline{x_1} \cdot f(0, x_2, x_3, \dots, x_n)$

**Example**

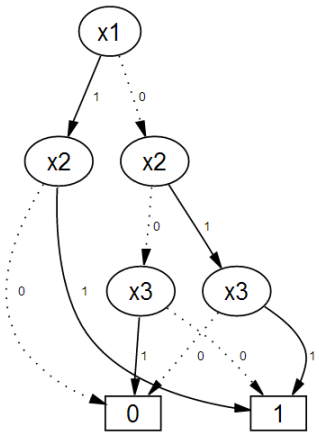
The Boolean function $f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\,x_2 + x_2\,x_3$ with the following truth table:

| 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, DataStructureType) → BddTree.
    - The function receives a Boolean function and returns the corresponding BDD tree representation for that Boolean function.
    - 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** which is the optimal tree (minimum height, minimum number of nodes and minimum number of leaves).
        ii. To extract the optimal 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: $f_s(x_1, x_2, x_3)$. $f_s$ can 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: $\{\{x_1, x_2, x_3\}, \{x_1, x_3, x_2\}, \{x_2, x_1, x_3\}, \{x_2, x_3, x_1\}, \{x_3, x_2, x_1\}, \{x_3, x_1, x_2\}\}$
    - The returned tree must be reduced by Rule 1. Read more about in BDD Introduction.

- Data structure type can be **map** (atom) or **record** (atom) (both methods should be implemented).

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.
3. **spec listOfLeaves**(BddTree) -> Res.
   - Returns list of pointers to leaves.
4. **spec reverseIteration**(LeafPtr) -> Res.
   - Input is pointer to leaf (one of leaves given in result list of step 3).
   - Returns list of nodes on shortest path to the root.

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

*Note: signatures of these functions must be exactly the same as specs. This assignment will be checked by a testing machine. 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
- First element of an operator tuple is an atom.

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

Will be represented as

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