Mid Semester Project

Functional Programming in Concurrent and Distributed Systems

Ron Kotlarsky

ID 204130538

Representation of Tree Elements

node: {x1, false node,true node}
leaf: {true}/{false}
tree of tautological expression: {true}

Description of functions

- findVars(List,Expression) – receives a boolean expression tuple (using the format specified in the assignment), and adds any atoms it finds to List using pattern matching. Returns a List of atoms containing duplicates which are deleted using remove_dups function.
- solveExp(Expression, varMap) – receives a boolean expression tuple, and solves it using a map of variables, returns true\false.
- BuildInitialTree(Expression, Map, List) – Receives a boolean expression, a map of vars which represents the values of parent vars decisions, and a List of variables which represents the order of variables in the tree. Once the function reaches a leaf it calls solveExp which solves the expression using the Map of decisions the leaf receives. Function first creates Left and Right children recursively, and if Left=:=Right, the are joined and the current node the function handles becomes the value of Left\Right.
- getTreeParams(Tree, Params, Depth) – Receives a tree and parameter map ({treeHeight=>0,numOfNodes=>0,numOfLeafs=>0} at first), and calculates the requires parameters. Returns the updated parameter map.
- getOptimal(Expression,Permutations,Ordering,CurrentMinOrdering,OptimalTuple) – Receives permutations of variables and returns optimal tree according to Ordering. In case there's more than one optimal tree the function returns the first optimal tree it receives.
- TreeSearch(Tree,VariableMap) – Searches the given tree for the solution to VariableMap.

Solving the function f(x1,x2,x3,x4)

The following call is made to solve f, in the most efficient way by treeHeight (for example):

```
midproject_204130538:exp_to_bdd({'or',{{'and',{x1,{'and',{x3,
{'not',x2}}}}},{'or',{{'not',{'and',{x1,{'and',{{'not',x3},{'or',{x2,
{'not',x4}}}}}}}},{'not',{'and',{x4,x1}}}}}}},treeHeight).
```

The following call is made to get value of f with given x1,x2,x3,x4:

```
midproject_204130538:solve_bdd(Q,[{x1,1},{x2,1},{x3,1},{x4,1}]).
1
```

Printing all the possible permutations:
Note: treeHeight starts at 0, numOfNodes does include leaves, tree with only a root is considered a leaf.

```
 {{x4,{true},{x1,{true},{x2,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x4,{true},{x1,{true},{x3,{x2,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x4,{true},{x2,{true},{x1,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x4,{true},{x2,{true},{x3,{x1,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x4,{true},{x3,{x1,{true},{x2,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x4,{true},{x3,{x2,{true},{x1,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x4,{true},{x2,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x4,{true},{x3,{x2,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x2,{true},{x4,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x2,{true},{x3,{x4,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x3,{x4,{true},{x2,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x1,{true},{x3,{x2,{true},{x4,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x4,{true},{x1,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x4,{true},{x3,{x1,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x1,{true},{x4,{true},{x3,{false},{true}}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x1,{true},{x3,{x4,{true},{false}},{true}}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x3,{x4,{true},{x1,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x2,{true},{x3,{x1,{true},{x4,{true},{false}}},{true}}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x4,{true},{x1,{true},{x2,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x4,{true},{x2,{true},{x1,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x1,{true},{x4,{true},{x2,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x1,{true},{x2,{true},{x4,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x2,{true},{x4,{true},{x1,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}

{{x3,{x2,{true},{x1,{true},{x4,{true},{false}}}},{true}},
 #{numOfLeafs => 5,numOfNodes => 9,treeHeight => 4}}
```

In the case of the function we were asked to solve, all trees have the same efficiency! This is caused since the function equals ‘false’ only once, which means it is not possible to eliminate any variable since we need to consider the case the function receives ‘false’.

Execution Time

```
76> midproj:exp_to_bdd(Tot,treeHeight).
Execution Time (millis) 1

{x4,{true},{x1,{true},{x2,{true},{x3,{false},{true}}}}}
```

The Execution time for comparing the most efficient Tree is 1 millisecond, which is the smallest resolution I chose to work with.

Conclusions

The problem of ROBDD tree is of NP order since it requires building n! trees, and is done quite by “brute force”, excluding several possible optimizations. This work has taught me many things and it was interesting to implement all the things I have learned in sequential Erlang.

Comparison of sequential and parallel

Parallel Erlang would have been able to build several BDD trees in a parallel way. Since the built BDD trees do not depend on each other, it is possible to calculate all possible ROBDD’s in a parallel manner, and once all trees are calculated to decide which one is the optimal. So in conclusion, parallel Erlang would have been much more efficient in solving this problem.