DJBDD

Índice

Introduction

This document explains the Binary Decision Diagram library developed in Java 7.

Me, Diego J. Romero is the author of this BDD package.

This document is not the final version of the develoment manual. It serves me as a TODO list, journal and diary.

Highlights

We don't use the reduce operation. Our BDD is reduced when constructed. So we enforce the properties of uniqueness and non-redundantness of the vertices when constructing the BDD.

Data structures

Vertex

The vertex is composed of the following attributes:

- **`index`**: an unique index for this vertex. The way we get the indices, they are not repeated anytime.
- **`variable`**: an index to the variable that contains this vertex.
- **`low`**: a reference to the low vertex.
- **`high`**: a reference to the high vertex.

BDD Tree

The basic structure we use is a vertex tree with some enhancements.

This tree contains three hash tables that do not reference any vertex. That is we use a new feature of Java 7 called weak references. This way we do not have to make some explicit garbage collection, we can rest assure the Java garbage collection will do its work, and we only should we be wary of the size of the tables, compacting them to erase the empty values.

Table T

The first hash table contains a hash whose keys are the indices of each vertex and its values, references to its vertices.

It is used to keep count of the vertices and as vertex cache, easing the transversal of the tree.

Table U

The uniqueness table. This hash has as key a concatenation of the attributes that makes unique one vertex, that is:

- `variable`: variable index of the vertex.
- `lowid`: id of the low descendant. If our vertex is a leaf, thus making the high descendant a null, we replace it with a constant.
- `highid`: id of the high descendant. If our vertex is a leaf, thus making the high descendant a null, we replace it with a constant.

This table is used to enforce the uniqueness of each vertex when adding new vertices in the construction of the BDD or when using the apply operation.

Table V

Contains the vertices in groups by variables. That is, the key is the index variable and the value is a set of vertices.

At this moment this structure is not used for anything. We have some plans to use it in a heuristic variable ordering algorithm.

Operations

Construction

The construction uses the apply operation and a grammar parser[1] which is faster than the recursive process. This algorithm constructs the BDD making use of the apply operation between the internal formulas and variables of the formula. The basic BDDs that contains one variable (or a complemented variable) will be constructed with a private static factory with the name of `BDD.factoryFromVarible`.

If you want to change that, set `BDD.USE_APPLY_IN_CREATION` to `false`.

Apply

1 This parser uses Antlr3 (http://www.antlr.org/wiki/display/ANTLR3/ANTLR+3+Wiki+Home)

Garbage collection

The garbage collection can be called using `BDD.gc`. Note that this garbage collection compacts table T, erasing the pairs `<index,WeakReference<Vertex>>` where the vertex has been erased.

Open tasks

There are some tasks that have not been finished.

Dynamic variable ordering

First, we have to implement some kind of dynamic variable reduction algorithm. Currently, our implementation does not work and your advised to not use it.

We are on the way to implement the Sifting Algorithm explained by Ruddel in [Rud93].

Parallel algorithm

One of my aims is making this implementation run on shared memory environments.

Bibliography

[1] Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, Randal E. Bryant. Carnegie Mellon University.

[2] Binary Decision Diagrams. Fabio SOMENZI. Department of Electrical and Computer Engineering. University of Colorado at Boulder.

[3] Efficient implementation of a BDD package, Karl S. Brace, Richard L. Rudell, Randal E. Bryant.

[4] Implementation of an Efficient Parallel BDD Package. Tony Stornetta, Forrest Brewer.

[Rud93] *Dynamic variable* ordering for ordered binary decision diagrams. Richard L. Rudell 1993.