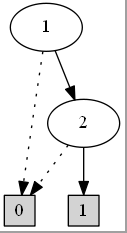
 BDD v1 = B.ithVar(1);

v1.printDot();

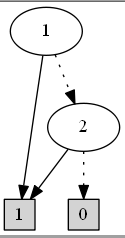


BDD v1 = B.ithVar(1);

BDD v2 = B.ithVar(2);

BDD a = v1.and(v2);

a.printDot();

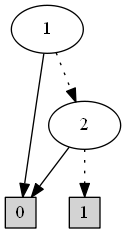


BDD v1 = B.ithVar(1);

BDD v2 = B.ithVar(2);

BDD a = v1.or(v2);

a.printDot();



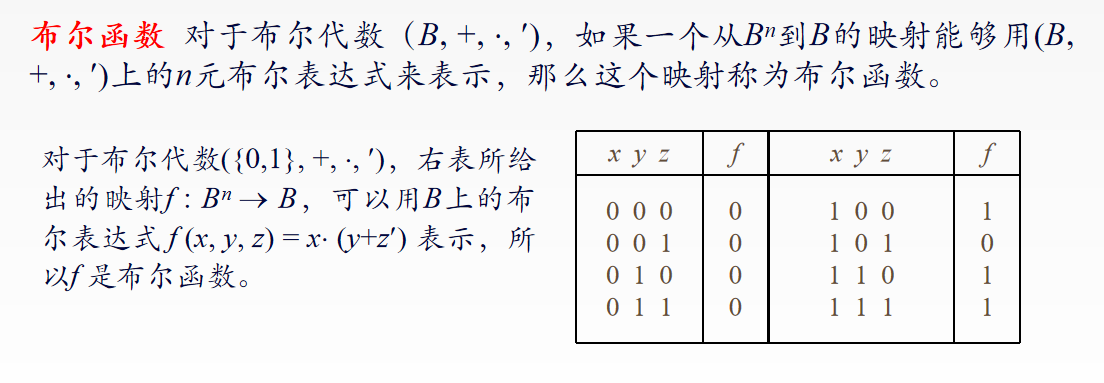
BDD v1 = B.ithVar(1);

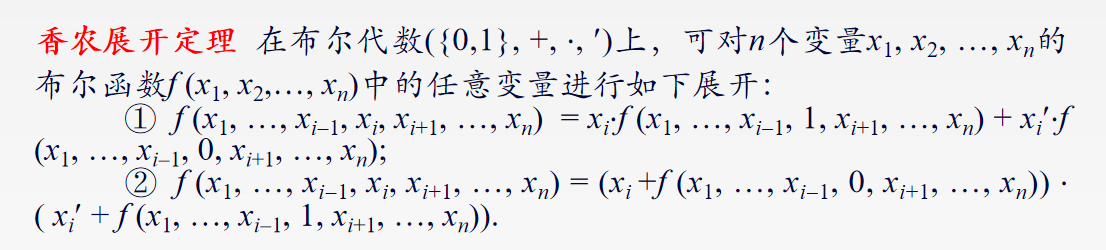
BDD v2 = B.ithVar(2);

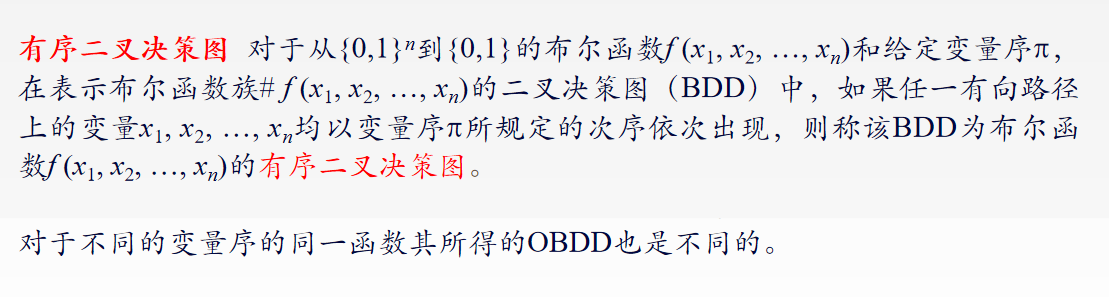
BDD a = v1.or(v2).not();

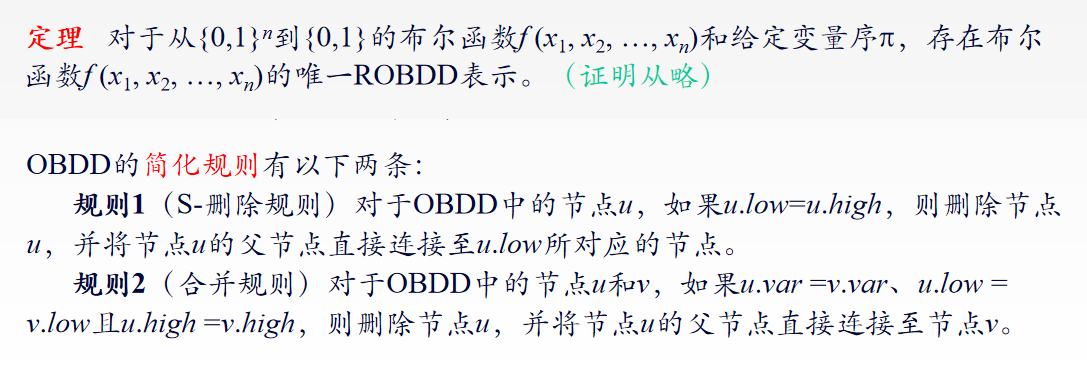
a.printDot();

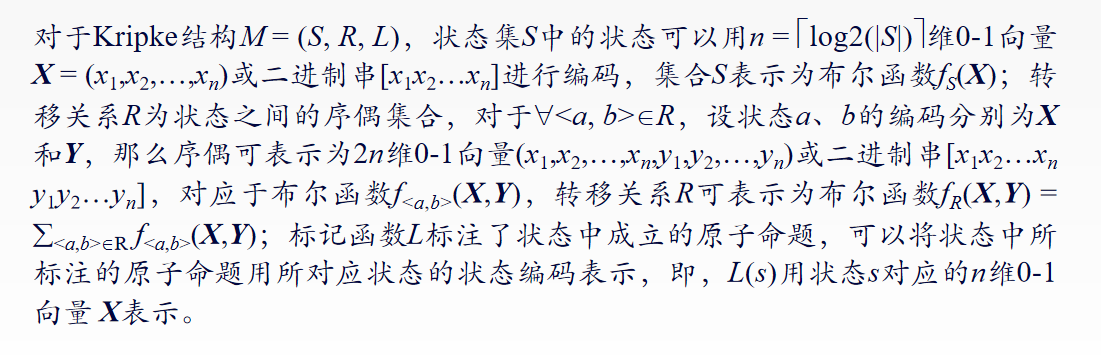
注：not()并不一定会将最下面一层的1和0交换位置，也可能会通过交换某些边来实现。可以通过 BDD a = v1.xor(v2) 和 BDD a = v1.xor(v2).not() 来验证。

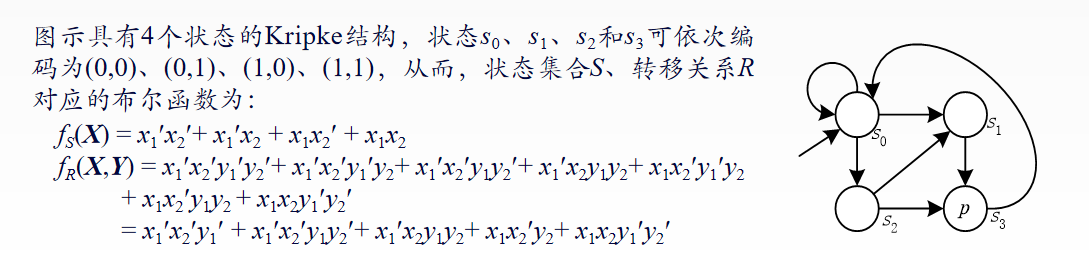


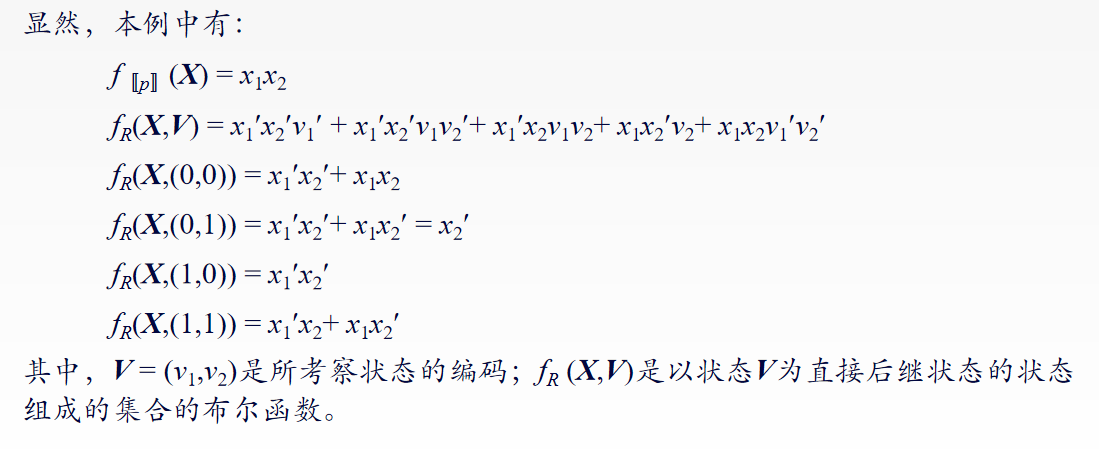






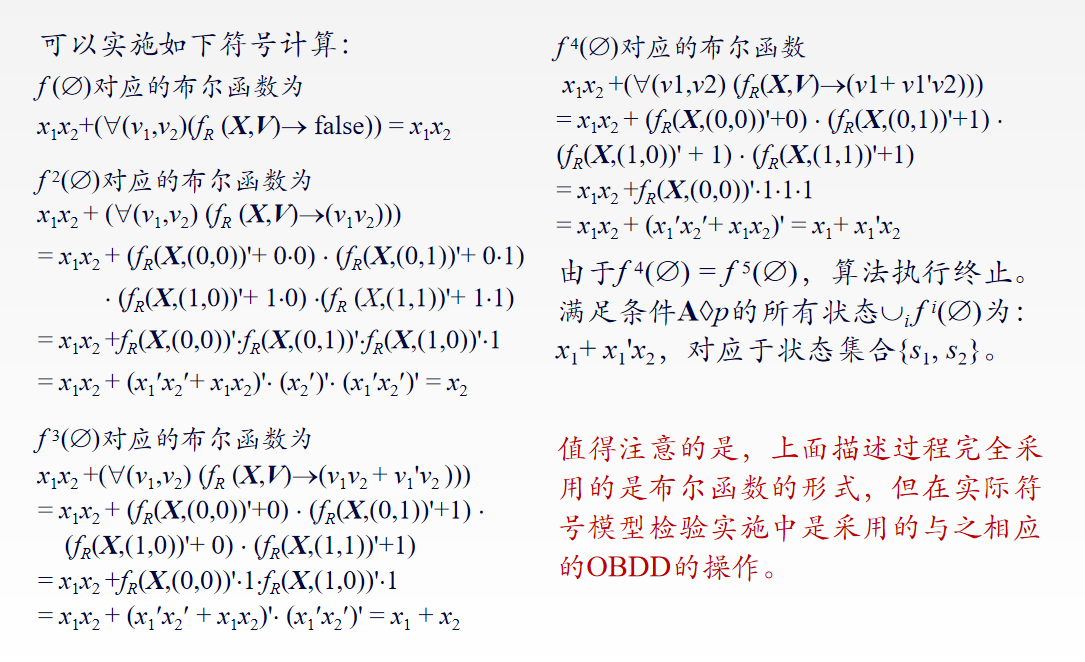






fR (X,V)：也就是说这个集合包含着一些状态，这些状态的下一个状态就是V。

（这些状态是由X表示的）



重要且常用的函数：

**[printSet()](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDD.html" \l "printSet()) - Method in class net.sf.javabdd.[BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd)**

Prints the set of truth assignments specified by this BDD.

以文本格式输出从初始节点到**终点1**的每条路径

**[printTable(BDD)](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDDFactory.html" \l "printTable(net.sf.javabdd.BDD)) - Method in class net.sf.javabdd.[BDDFactory](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDDFactory.html" \o "class in net.sf.javabdd)**

Prints the node table entries used by a BDD.

自底向上输出BDD各节点直接的连接情况

例子：BDDFactory.printTable(dBdd);

**[apply(BDD, BDDFactory.BDDOp)](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDD.html" \l "apply(net.sf.javabdd.BDD, net.sf.javabdd.BDDFactory.BDDOp)) - Method in class net.sf.javabdd.[BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/./net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd)**

Returns the result of applying the binary operator opr to the two BDDs.

对两个BDD进行某种二元操作，返回值为结果BDD。二元操作是通过枚举类定义的，一共有十种，如下所示。

**Class BDDFactory.BDDOp：**

Enumeration class for binary operations on BDDs. Use the static fields in BDDFactory to access the different binary operations.

例子：BDD a = v1.apply(v2, BDDFactory.imp);

BDDFactory.imp : P → Q 仅当P真Q假时才为0

BDDFactory.invimp: Q → P 仅当P假Q真时才为0，相当于反过来的imp

BDDFactory.biimp: P ↔ Q 同或，相同就是1，不同就是0

BDDFactory.xor: 异或，相同就是0，不同就是1

BDDFactory.and: P∧Q 合取

BDDFactory.nand: NOT(P∧Q) 与非，结果是与的相反情况

BDDFactory.or: P∨Q 析取

BDDFactory.nor: NOT(P∨Q) 或非，结果是或的相反情况

BDDFactory.diff: P∧NOT(Q)

BDDFactory.less: (NOT)P∧Q

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) ite([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) thenBDD,[BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) elseBDD)**

if-then-else operator.Compare to bdd\_ite.进行if-then-else操作。

例子：BDD a = v1.ite(v2,v3);

**public [BDDPairing](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDDPairing.html" \o "class in net.sf.javabdd)** **makePair(int oldvar, [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) newvar)**

Make a new pairing that maps from one variable to another BDD.

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) replace([BDDPairing](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDDPairing.html" \o "class in net.sf.javabdd) pair)**

Returns a BDD where all variables are replaced with the variables defined by pair. Each entry in pair consists of a old and a new variable. Whenever the old variable is found in this BDD then a new node with the new variable is inserted instead.

替换操作将某序号的BDD换成一个新的BDD。makePair函数还有很多其他的传参方式。

两个函数结合使用的例子：

BDDPairing aPairing = B.makePair(2, v4);

BDD cBdd = a.replace(aPairing);

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd)** **restrictWith([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) var)**

Mutates this BDD to restrict a set of variables to constant values. Restricts the variables in this BDD to constant true if they are included in their positive form in var, and constant false if they are included in their negative form. The "that" BDD is consumed, and can no longer be used.

Note that this is quite different than Coudert and Madre's restrict function.

Compare to bdd\_restrict and bdd\_delref.

进行restrict操作，相当于在一个BDD的值是已知的情况下，画出整个图像。

例子：

g.restrictWith(v1);

//added line - use 'g.restrictWith(v1.not());' for negative restriction

注：restrict([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) var)方法和该方法类似，只是需要申请一个新的BDD，而restrictWith方法不需要，直接在原来BDD的基础上进行。

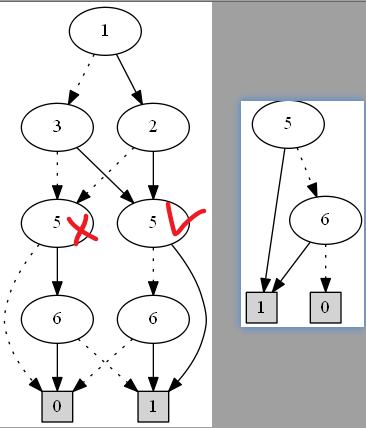
例子：

BDD dBdd1 = cBdd.restrict(v4.not());

dBdd1.printDot();

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) simplify([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) d)**

Coudert and Madre's restrict function. Tries to simplify the BDD f by restricting it to the domain covered by d. No checks are done to see if the result is actually smaller than the input. This can be done by the user with a call to nodeCount().

Compare to bdd\_simplify.

例子：

BDD a = v1.ite(v2,v3);

BDD b = v6.ite(a,v5);

BDD c = b.simplify(a);

输出b和c的图像后，可以发现c把b中的某些节点去掉了，

这些节点是a中节点的集合。而且和上个方法的例子类似，

这里的c保留的是之前节点为1的路径，如果想要保留走

0那一边对应的路径，需要写成b.simplify(a.not());

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) compose([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) g, int var)**

Functional composition. Substitutes the variable var with the BDD that in this BDD: result = f[g/var].

Compare to bdd\_compose.

这个函数感觉比较重要但是不太懂。。。经常出现空指针错误

BDD.compose() may be used instead of BDD.replace() but is not as efficient when gi is a single variable, the same applies to BDD.restrict().

BDD.compose()和BDD.restrict() 都可以用来代替BDD.replace()，但是当要替换的元素只有一个变量时，效率并没有使用BDD.replace()那么高。

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd)** **costrain([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) that)**

Generalized cofactor. Computes the generalized cofactor of this BDD with respect to the given BDD.

Compare to bdd\_constrain.

相当于把参数中的BDD的值约束为1或0。和restrict()函数很像。

例子：

BDD cBdd = b.constrain(v2.not());　//这里被约束为0

**public abstract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) exist([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) var)**

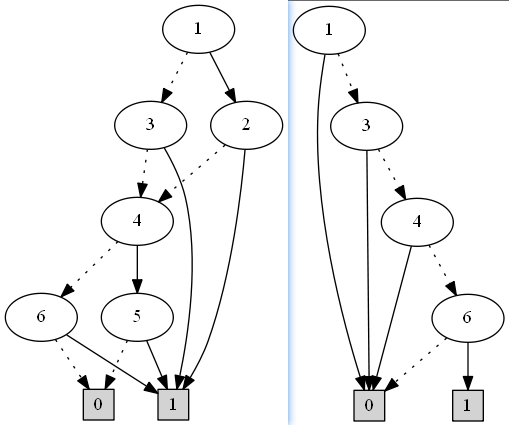
Existential quantification of variables. Removes all occurrences of this BDD in variables in the set var by existential quantification.

Compare to bdd\_exist.

从结果上看，是从已有的BDD中直接删除了var这个BDD节点。

**public ab****stract [BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd)** **satOne()**

Finds one satisfying variable assignment. Finds a BDD with at most one variable at each level. The new BDD implies this BDD and is not false unless this BDD is false.

Compare to bdd\_satone.

找到一条路径并用BDD表示出来。

在每一层最多包含一个BDD。

如图所示，左边的是BDD原图，

右边的是使用该函数生成的BDD图。

有个类似的函数，用来统计到1的边的数目，但是结果和预期不一样

public abstract double satCount()

Calculates the number of satisfying variable assignments.

Allsat()函数的例子，用来得到所有到1的边的路径，但是结果和预期不一样，没有什么用，只能用来统计数目

例子：

List<byte[]> lines = test.allsat();

for (byte[] object : lines) {

for (int i = 0; i < object.length; i++) {

System.out.print(i+" ");

}System.out.println();

}

**public abstract boolean equals([BDD](C:/Users/cde/Desktop/%E6%AF%95%E8%AE%BE%E7%9B%B8%E5%85%B3/javabdd_1.0b2/apidocs/net/sf/javabdd/../../../net/sf/javabdd/BDD.html" \o "class in net.sf.javabdd) that)**

Returns true if this BDD equals that BDD, false otherwise.判断两个BDD是否相等。

例子：

BDD f = v1.apply(v2, BDDFactory.biimp);

BDD g = v1.and(v2).or(v1.not().and(v2.not()));

System.out.println(f.equals(g));

**public int level()**

Gets the level of this BDD.

**public abstract int var()**

Gets the variable labeling the BDD.

这两个函数都返回BDD根节点的变量的序号。

**public abstract int[] varProfile()**

Counts the number of times each variable occurs in this BDD. The result is stored and returned in an integer array where the i'th position stores the number of times the i'th printing variable occurred in the BDD.

Compare to bdd\_varprofile.

返回nodeTable中的每个BDD出现的次数

public [BigInteger](http://java.sun.com/j2se/1.4.2/docs/api/java/math/BigInteger.html" \o "class or interface in java.math)[] scanAllVar()

Finds one satisfying assignment in this BDD of all the defined BDDDomain's. Each value is stored in an array which is returned. The size of this array is exactly the number of BDDDomain's defined.

有的时候，Java BDD中的一些方法的返回值也是BDD，比如support()函数，返回这个BDD中用到的所有节点。这说明结果被包含在这个BDD中了。此时如果用printDot()函数输出结果图，虽然也能看到一些信息，但是并不合适，最后使用printSet()等函数，会得到更清晰的结果。

**BDDdomain类**

vars

public int[] vars()

Returns an integer array containing the indices of the BDD variables used to define this finite domain. Compare to fdd\_vars.

返回domain中的变量