HOKKAIDO UNIVERSITY

Title	BDD/ZDDパッケージ講習会
Author(s)	湊, 真一
Citation	2010年度科学技術振興機構ERATO湊離散構造処理系プロジェクト講究録. p.249-280.
Issue Date	2011-06
Doc URL	http://hdl.handle.net/2115/48423
Type	conference presentation
Note	ERATOセミナ2010 : No.39. 2011年3月22-23日
File Information	39_all.pdf

Instructions for use

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

ERATO セミナ 2010 - No. 39

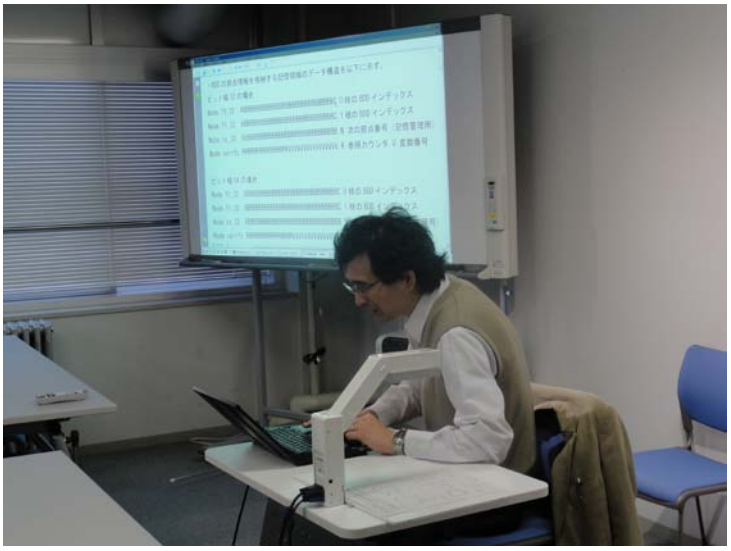
BDD/ZDD パッケージ講習会

湊 真一
北海道大学情報科学研究科 / ERATO 研究総括

2011 年 3 月 22, 23 日

概要

研究総括の湊が実装して使用している北大版 BDD/ZDD パッケージのソースの構成やデータ構造などを解説します。パッケージの詳細に興味のある方（および実際に使いそうな学生さんなど）は、どうぞ聴講してください。第 1 回は、BDD/ZDD のデータ構造の概要の解説（湊が大学院の講義で用いている資料を再構成）が中心です。第 2 回は、実際のソースコード（C および C++）とドキュメントを見ながら、BDD/ZDD を表す構造体のフォーマットや、内部の関数呼び出しの構成を解説します。また、パッケージのインストールの方法と、簡単な例題のコンパイル等の実演も行います。



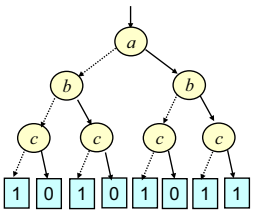
BDD/ZDDパッケージ講習会

北海道大学 / JST ERATO
湊　真一

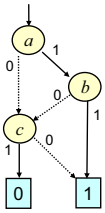
今回の内容　BDD処理系の実装技術

- BDDデータ構造の復習
- データ構造
  - 節点の表現、テーブル上の表現
- 節テーブルによる一意性の保証
  - ハッシュテーブルの実装
- 論理演算アルゴリズム
  - 再帰的アルゴリズム、シャノンの展開
  - 演算例
  - 演算キャッシュの実装と計算時間
  - 否定演算と否定エッジ
  - 代入演算と計算時間
- 記憶管理
  - 参照カウンタとガベジコレクション、演算キャッシュの初期化
  - 記憶領域の動的拡張

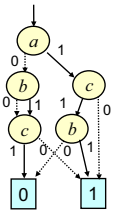
BDD（Binary Decision Diagram）（二分決定グラフ）とは



真理値表と等価なBDD
(Binary Decision Tree)



既約な順序付きBDD
(Reduced Ordered BDD)

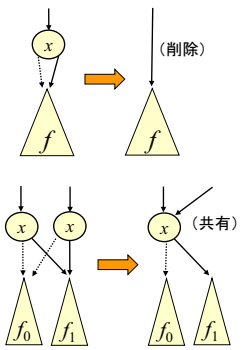


既約でも順序付き
でもないBDD
(Unordered BDD)

ROBDD (Reduced Ordered BDD)

- 同じ論理を表すBDDは複数存在
- 重要な性質を持つのは既約な順序付きBDD(ROBDD)
  - **以後、特に断らない限り、ROBDDのことを単にBDDと呼ぶ。**
- 順序付きBDD：
  - 変数に**全順序関係**が定義されている
  - 根(root)から定数節点に至るすべてのパスについて変数の出現順序が、全順序関係に矛盾しない
- 既約なBDD
  - BDDの2つの簡約化規則がこれ以上適用できなくなるまで適用されている形

BDDの簡約化規則



(a) 冗長な節点を全て削除
(b) 等価な節点を全て共有

↓

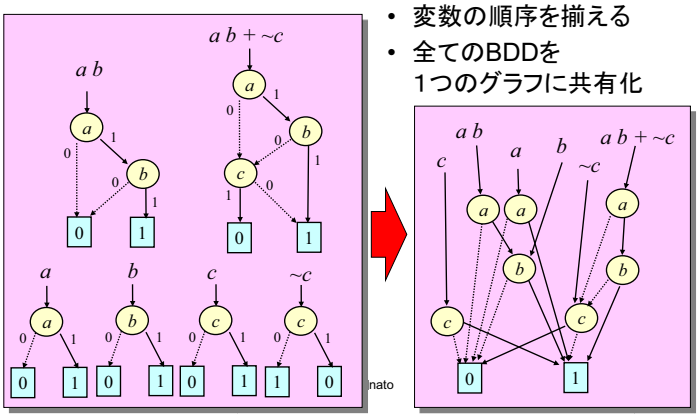
既約なBDDが得られる

参考：
(b)の規則だけを可能な限り適用した形を「準既約」(Quasi-reduced)なBDDと呼ぶこともある。

BDDの特徴

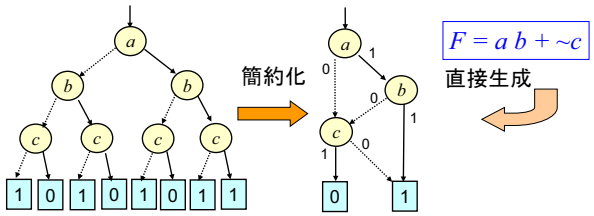
- **論理関数に対してグラフの形が一意に定まる。**
  - 等価性判定が非常に容易
- **多くの実用的な論理関数がコンパクトに表現できる。**
  - パリティ関数や加減算回路も効率よく表現
  - 性質の良い関数では数百入力まで扱える
- **論理関数同士の演算が、グラフのサイズにほぼ比例する計算時間で実行できる。**
  - 否定演算も容易
- グラフのサイズが小さくならない場合もある。
  - 乗算回路のBDDは指数サイズ
- 変数の順序づけが悪いとグラフが大きくなる。
  - 比較的良い順序づけを得る方法がいくつか実用化（厳密最小化はNP完全問題）

複数のBDDの共有化(Shared BDD)



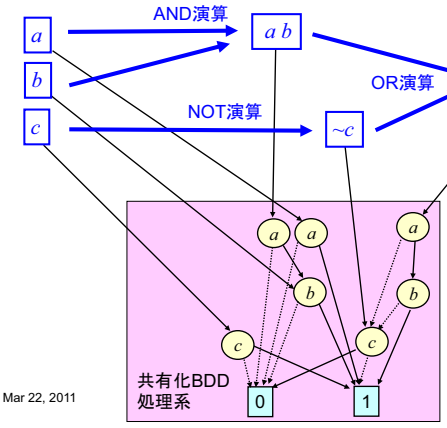
- 変数の順序を揃える
- 全てのBDDを1つのグラフに共有化

BDDの生成アルゴリズム



- 真理値表に対応する二分木を簡約化する方法では、常に指数オーダの記憶量と処理時間がかかってしまう。
  → 実用的には、論理式からBDDを直接生成するアルゴリズム[Bryant86]を用いる

共有化BDDでの論理演算処理

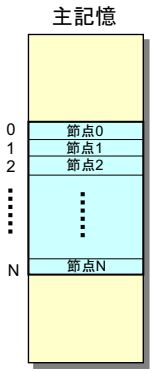


- 論理演算の実行に従い、共有化BDDが成長していく
- 共有化BDD処理系では、論理の一致判定はポインタの一致判定のみで行える(定数時間)

BDDパッケージ

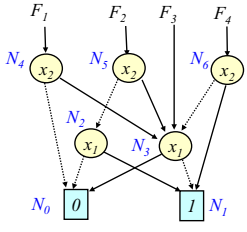
- BDD処理系は世界各地の研究機関で1990年頃から盛んに開発された。
  - BDDパッケージとして無料配布されている物もいくつかある。
- 多くの場合、CまたはC++のライブラリとして提供されている。
  - **BDDへのポインタを引数としてライブラリ関数を呼び出すと、メモリ上にBDDが生成され、新しいBDDへのポインタが返り値として戻ってくる。**
  - ユーザはBDDの論理演算を呼び出すメインプログラムを書き、BDDパッケージをリンクしてコンパイルすると、BDD処理アプリケーションを作ることができる。
- 複数のBDDを統一的に扱う共有化BDDの技法が広く用いられている
  - **これ以降、原則として共有化BDD処理系について解説する。**

BDDの計算機上での内部表現



- BDD処理系では全てのBDDの節点を、計算機の主記憶(メモリ)に保持し、統一的に管理している。
- 各節点は**(入力変数番号、0枝、1枝)**の3つの属性データと、グラフの管理用のポインタやカウンタなどの付属データを持つ。
- 典型的な実装では、節点を格納する記憶領域は0番地からの**連続する領域**に、テーブルとして確保している。
  - 0枝、1枝はそれぞれの行き先の節点の番地(インデックス)を格納。
  - 変数番号は自然数を使い、最下位を1として上位ほど大きな番号で表示。(処理系によっては逆に上位を1にしている場合もあるが、上下の区別がつけばどんな番号付けでもよい。)

BDD節テーブルの実装例



$F_1 = \sim x_1\, x_2$
$F_2 = EXOR(x_1, x_2)$
$F_3 = \sim x_1$
$F_4 = \sim x_1 + x_2$

	変数	0枝	1枝	
N_0	-	-	-	← 0定数節点
N_1	-	-	-	← 1定数節点
N_2	x_1	N_0	N_1	
N_3	x_1	N_1	N_0	← $F_3 (= \sim x_2)$
N_4	x_2	N_0	N_3	← $F_1 (= \sim x_1\, x_2)$
N_5	x_2	N_2	N_3	← $F_2 = EXOR(x_1, x_2)$
N_6	x_2	N_3	N_1	← $F_4 (= \sim x_1 + x_2)$

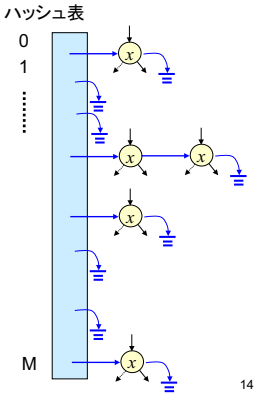
- 全ての論理関数は**グラフの根の節点の番地(インデックス)で識別される。**
- インデックスは、最大節点数をMとすると$\log M$ビットの記憶量を要する。
  - 通常は1ワード(多くの場合32ビット)の整数を使って表す。(約40億個まで識別可能)

節テーブルによる一意性の保証

- BDD処理系では、共有可能な部分グラフは必ず共有されていなければならない。
  - 等価な節点が重複して存在してはならない。
  - 変数番号、0枝、1枝の3つの属性が全て一致する節点がすでに存在していたら、新しい節点を作らずに、既存の節点の番地のみを返すようにする。
- 節点の一致判定には、ハッシュテーブルの技法を用いる。
  - 全ての節点は、(変数番号、0枝の番地、1枝の番地)の3つの数値をキーとするハッシュテーブルに登録しておく
  - **新たな節点を生成する前に必ずハッシュテーブルを検査して、等価な節点があれば共有する。重複する節点は一切作らない。**
  - ハッシュテーブルの検査は**定数時間。**(節点数が主記憶に収まっている限り)

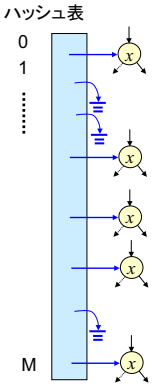
ハッシュテーブルの構成例(1)

- 外部ハッシュ方式
  - 3つの属性を適当に組合せて、ほぼランダムに散らばるハッシュ値を作り、その番地にBDD節点へのポインタ(インデックス)を格納
  - 運悪くハッシュ値が衝突した場合は、リストを作ってつなげて格納する。
  - ハッシュ表の配列に加えて、リストを作るためのポインタの記憶領域(節点ごとに1ワード)が必要
  - ハッシュ表のサイズは最大節点数と同程度あれば良好に動作(平均アクセス時間＝定数サイクル)

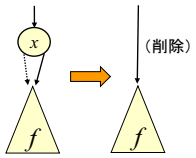


ハッシュテーブルの構成例(2)

- 内部ハッシュ方式
  - ハッシュが衝突した場合に、次の空いている番地に記録。
  - 参照する際には、空欄が現れるまで順番に番地を増やしてチェックする。
  - ハッシュ表の配列のみで格納可能
  - 良好に動作するためのハッシュ表のサイズは最大節点数の**2倍程度**必要。
  - 良好に動作させるために必要な記憶量は**外部ハッシュ方式とほとんど同じ。**



冗長な節点の生成抑制

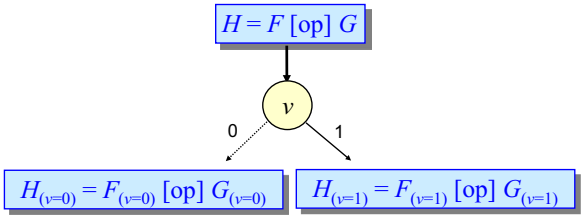


- 0枝と1枝が同じ部分グラフを指している場合は、新しい節点を作らずに、部分グラフをそのまま返す。
- BDD処理系では等価な節点が複数個存在しないことが保証されているので、**0枝と1枝の番地を比較するだけで**、節点が冗長かどうかが判定できる。

手続き *GetNode(v, F0, F1)*

- これまで説明した節テーブルの技法を1つの手続きにまとめたもの
  - BDD処理系の中で**最も基礎的な手続き**
  - 種々の演算の過程で、必要な節点を得るために呼び出される。
- *GetNode*の動作(仕様):
  引数として(変数番号:v, 0枝の番地:$F0$, 1枝の番地:$F1$)を与えられたときに、
  - $F0$と$F1$が等しければ$F0$をそのまま返す。
  - 節テーブルを検査して、等価な節点が見つかれば、その番地を返す。
  - 等価な節点がなければ新しい節点を作ってその番地を返す。
  - 新しい節点を作ろうとして最大節点数を超えた場合には**エラーを返す。**

二項論理演算アルゴリズム

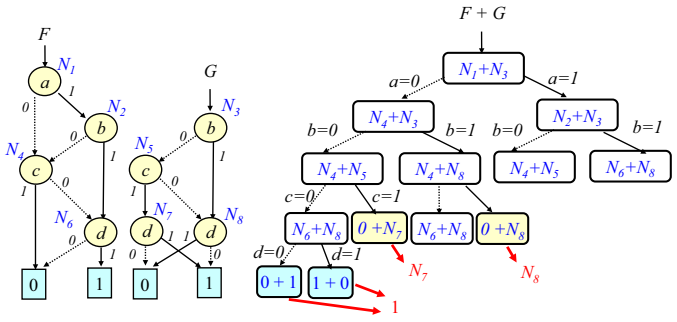


- ある変数vに0,1を代入して再帰的に展開
- 全ての変数を展開すると自明な演算になる
  (OR演算の場合) $F+1=1$, $F+0=F$, $F+F=F$ …
- 各演算結果をBDDに組み上げる

二項論理演算手続き *APPLY(op, F, G)*

1. *F,G*のいずれかが定数のとき、および*F=G*のとき
   - 演算子の種類 *op* に応じた演算結果の節点の番地を返す。
     （例）*F* [AND] *0* → *0*,　*F* [OR] *F* → *F*,　*F* [EXOR] *0* → *F* など
2. 両者の最上位変数 *F.v*と*G.v* が同じとき
   - H_0←*APPLY(op,*F_0*,*G_0*)*, H_1←*APPLY(op,*F_1*,*G_1*)*を再帰呼出し
   - H_0=H_1であればH_0を返す。そうでなければ、*GetNode(F.v,* H_0*,* H_1*)*を呼出し、得られた節点の番地を返す。
3. *F.v* が *G.v* よりも上位のとき
   - H_0←*APPLY(op,*F_0*,G)*, H_1←*APPLY(op,*F_1*,G)*を再帰呼出し
   - H_0=H_1であればH_0を返す。そうでなければ、*GetNode(F.v,* H_0*,* H_1*)*を呼出し、得られた節点の番地を返す。
4. *F.v* が *G.v* よりも下位のとき
   - *F*と*G*を入れ替えて、3.と同様に処理

二項論理演算の実行例



演算キャッシュによる高速化

- *APPLY*演算の再帰呼出しは二分木状になる
  - 通常の逐次処理系では二分木を深さ優先順にたどりながら実行する。
  - 元のBDDに共有があるため、途中で**同じ節点の組が多数現れる**ことがある。
- 過去に演算を行った節点の組とその演算結果を記録する「演算キャッシュ」を用意すると、処理を高速化できる。
  - *APPLY*を実行するときに、同じ*(op,F,G)*の組が演算キャッシュに登録されていれば、再帰処理を打ち切り即座に結果を返せる。
  - キャッシュがすべてヒットすれば、*F,G,H*の節点数の総和にほぼ比例する時間で*APPLY*演算を実行できる。

演算キャッシュのデータ構造の例

op	*F*	*G*	*H*
OR	N_1	N_3	N_8
-	-	-	-
-	-	-	-
AND	N_4	N_0	N_0
AND	N_7	N_8	N_3
-	-	-	-
EXOR	N_3	N_5	N_4

- (*op, F, G*)の組をキーとするハッシュ表を作り高速に検索
  - 1エントリー当たり3.5ワード程度
- 全ての演算の組を記録しようとすると表が大きくなり過ぎる。
  - 最近の演算だけを記録する「キャッシュ」形式とする。
  - ハッシュ値が衝突した場合は、後からのデータを上書き。
  - 過去のデータが失われた場合、冗長な計算が増えて遅くなるが、結果の正しさには影響しない。

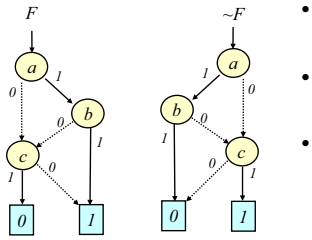
演算キャッシュのサイズと処理速度

- 演算キャッシュのサイズが不十分だと、無駄な演算が多くなり急激に速度が低下する。
  - 多くの場合、実験的に最適なサイズを決めている。（例えば最大節点数の1/4のエントリー数）
- 演算キャッシュがヒットする確率を上げる工夫：
  - 自明な演算（例えば定数を含む場合など）は登録しない。
  - 可換な演算（例えば *F+G = G+F*）ならば、どちらか一方のみ登録する。
  - 否定枝が使える場合は、*AND(F,G) = ~ OR(~F, ~G)*の変換式を用いて、演算子の種類を減らす。
  - 枝が2本以上集中している節点のみ記録する。

二項論理演算の計算時間

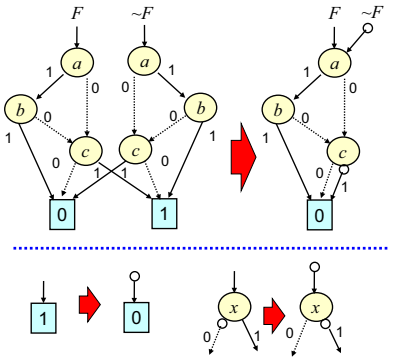
- グラフ *F* の節点数を $|F|$ と表すと、$H = F$ [op] G の節点数は、最悪の場合、$|H| = |F| \times |G|$ となる。
- 論理演算に要する計算時間は何も工夫しなければ、入力変数を *n* とすると $O(2^n)$となる。
  - しかし、ハッシュテーブルなどの効率化技法を使うことにより、$O(|F| + |G| + |H|)$ で抑えられる。
    - $|F|, |G|$ が大きくても $|H|$ が小さくなる場合がある。その逆もある。
  - グラフの節点数が小さくなるような場合であれば、*n*が大きくても劇的に高速に論理演算を行うことができる。
  - グラフのサイズが指数的に大きくなる場合（乗算器の例など）では、BDDの効果はほとんどない。

否定演算アルゴリズム



- BDDをコピーして、0と1の定数節点を交換するだけ
- グラフの節点数に比例する時間で計算可能。
- 次に述べる「否定枝」の技法を用いると**定数時間**で計算可能。

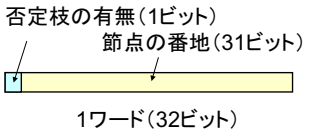
否定枝(Negative edge)



- 否定演算を表すマークを枝につけることで、否定同士の関係にあるBDDを共有化する技法
- 否定演算が**定数時間**で実行可能に
  - グラフの根の枝の否定枝をon/offするだけ
- グラフの一意性を保つため、否定枝の使用場所を制限している
  - 1の定数節点は使用しない
  - 0枝に否定枝を使用しない

否定枝の実装

- 否定枝の情報は、節点のインデックスに埋め込むことにより効率よく扱える。
  - インデックスを整数とみなすと、正負の符号に相当する。
  - BDDの否定演算は、インデックスの符号部分を反転するだけ（定数時間）
  - 否定枝も含めて、BDDの一致判定を1ワード整数の比較で行うことが可能。
  - ただし否定枝に1ビット割当てるため、処理系で扱える最大節点数が2^{32}個から2^{31}個に減る。
    - **普通はその限界より前に実メモリが足りなくなる。**



代入演算

- BDD F, 変数x, 定数値0（または1）が与えられたとき、変数xに定数値を代入したときのFを表すBDDを作り、その節点へのインデックスを返す演算
  - 代入変数 x がFの最上位変数$F.v$よりも上位にあるとき：
    → Fのインデックスをそのまま返す。
  - x が$F.v$と同じとき：
    → F_0（またはF_1）をそのまま返す（定数時間）。
  - x が$F.v$よりも下位にあるとき：
    F_0, F_1それぞれについて代入演算を再帰的に呼出し、その演算結果の節点を組み上げてBDDを作る。
- 演算キャッシュを効果的に使えば、Fの中で x より上位にある節点の個数に比例する計算時間で実行可能。

充足解探索

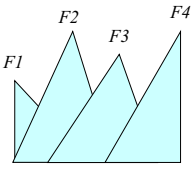
- ある論理関数が充足可能かどうか（恒偽関数でないかどうか）の判定は**ただちに可能。**
  - BDDを指すインデックスが0定数節点でなければよい
- 充足可能な場合に、充足解（論理を1にするための各変数の0,1の割当）を求めることも容易。
  - 0定数節点を指さない枝をたどっていけば、**必ず1定数節点に到達できる。**その経路が充足解を表す。
  - グラフの節点数ではなく、変数の数に比例する計算時間。
- 充足解の個数のカウント
  - 0枝側の解の個数と1枝側の解の個数を加えればよい（変数番号に飛び越しがあるときは個数を2倍する）
  - 演算キャッシュが機能すれば、**節点数に比例する時間**で計算可能

最適解探索・真理値表密度の計算

- 入力変数に1を代入するときのコスト（例えばxを1にすると100円、yを1にすると150円）が与えられているとき、コスト最小の充足解（最適解）を求める問題
  - （0枝側の最小コスト）と（1枝側の最小コスト＋最上位変数 vのコストの和）を比較して小さい方が全体の最小コスト。
  - 演算キャッシュに最小コストを保存すると、**節点数に比例する時間**で計算できる。
  - 各節点での最小コストがわかれば、コストが小さい方の枝を選んで降りていけば、その経路が最適解となる。
- 論理関数が1になる割合（真理値表密度）や確率も容易に求められる。
  - 0枝側と1枝側の真理値表密度の平均を取ればよい。
  - 演算キャッシュが機能すれば**節点数に比例する時間**で計算可能。
  - 各入力変数が1になる確率（例えばxは70%, yは40%）が与えられている場合でも同じ計算時間で実行できる。

BDD処理系の記憶管理

- 1節点あたりの記憶量(32ビットマシンの場合)
  - 節テーブル2.5ワード、ハッシュ表2ワード、演算キャッシュ3．５ワード×1/4 ……合計約5.5ワード(22バイト)
  - 1GBの主記憶で約3～4000万節点を格納できる。
  - 主記憶からあふれると急激に(100倍以上)遅くなる。
    - **ハッシュテーブルは極めてランダムなアクセスなので、ハードディスクのバッファやキャッシュが全く役に立たない。**
- 記憶領域の有効利用のための技法
  - **参照カウンタによる記憶管理:** 使用済みの節点を再利用したい
  - **動的な領域確保:** 小さいBDDも巨大BDDも効率よく扱いたい



参照カウンタによる記憶管理

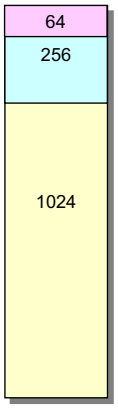
- 計算の途中結果のように、処理中に一時的に生成されて、二度と参照されないBDDが多数発生する。
  - 不要なBDDは解放して記憶の再利用を図ることが実用上不可欠。
  - 他のBDDと共有している節点は削除せず残す必要がある。
- 各節点ごとに参照されている枝の本数を保持する「参照カウンタ」の技法が多くのBDD処理系で用いられている。
  - 1節点あたり**1ワードの記憶量が余計に必要**。(合計で約26バイトとなる)
  - あるBDDが不要になったときは、根の節点の参照数を1減らし、0になれば実際に削除する。削除した場合は0枝1枝の参照数を減らし、再帰的に必要性を検査する。
  - BDDへのインデックスを勝手にコピーすることは許されない。
    - 参照カウンタの正当性を維持するため、処理系が提供する複製命令と削除命令を用いる。
    - C++やJavaを使えば、参照カウンタの管理をコンパイラに任せることができる。

演算キャッシュの初期化とガベジコレクション

- 節点を削除したとき、演算キャッシュにその節点のデータが残っていると、演算結果の正当性が失われてしまう。
  - 削除した節点を再利用すると、**インデックスの値は同じでも、以前と異なる論理関数になる。**
  - 削除した節点に関するデータをすべて探し出して除去することは困難なので、キャッシュ全体を初期化する等の対策が必要。
    →処理速度低下
- 処理速度低下を防ぐため、参照カウンタが0になっても削除せず温存しておき、どうしても足りなくなったときに一気にまとめて削除する技法(ガベジコレクション)が有効
  - 演算キャッシュの初期化回数を最小限に抑えられる
  - 削除してすぐまた必要になったときに即座に回復できる
  - ガベジコレクションは他の言語処理系でも使われる一般的な技法

記憶領域の動的拡張

- BDD処理系ではあらかじめ節点を記憶する領域を主記憶上に確保しておく必要がある。
  - どのくらい確保しておけばよいかは、**実行してみないとわからない。**
  - サイズが大きすぎると、実質的計算時間よりも初期化の時間の方が長くかかってしまう。
  - サイズが小さすぎるとあふれるかも知れない。
- 最初は小さいサイズにしておいて、足りなくなったら定数倍(例えば4倍)の領域を確保しなおす技法が有効。
  - 最後は主記憶のサイズで頭打ちにする。
  - 記憶を確保し直すために必要な時間は全体の計算時間の1/4以下で済む。



BDD処理系の実験結果の例

- 実用的な論理回路を集めたベンチマークで実験
  - 組合せ論理回路(ループや記憶素子を持たない回路)で、全ての出力端子および内部の途中計算部分の論理関数を同時にBDDで表現

	回路規模				
回路名	入力数	出力数	内部信号線数	BDD節点数	時間(秒)
sel8	12	2	29	78	0.3
enc8	9	4	31	56	0.3
adder8	18	9	65	119	0.4
adder16	33	17	129	239	0.7
mult4	8	8	97	524	0.5
mult8	16	16	418	66161	24.8
c432	36	7	203	131299	55.5
c499	41	32	275	69217	22.9
c880	60	26	464	54019	17.5
c1355	41	32	619	212196	89.9
c1908	33	25	938	72537	33.0
c5315	178	123	2608	60346	31.3

(計算時間はSun3, 24MByte)

BDD処理アルゴリズムのまとめ

- BDD処理の特長をまとめると、
  - 変数の展開順序を固定することにより、論理関数が内包する冗長性を自動的に抽出している。(一種の圧縮データ構造とも言える。)
  - **「同じ節点は2個持たない」**という効率化を徹底的に行う。
  - **「同じ計算を2度しない」**という効率化をできる限り行う。
- BDD処理の本質とは:
  - **ハッシュテーブルによる高速検索**
  - **ポインタによるリスト構造の操作**

「主記憶上の任意の番地のデータに定数時間でアクセス可能」という現在の計算機モデルの特長を最大限に活用している。

北大版BDDパッケージ

- 北大版BDD/ZDDパッケージ
  - 湊がNTTから北大に移籍した最初の年(2004年)に開発
  - NTT時代のパッケージを一から作り直した(特許・著作権問題なし)
- 現在、パッケージのソースは一般公開していない。
  - 法的には公開して問題ないが、質問や問合せに対応できない。
  - バージョンが一元管理できる体制が作れていない
  - まだドキュメントが日本語のみ
  - 現在でも開発が進んでいて、ときどき内部をいじっている
  - 個人的に付き合いのある人にだけソースを開示
  - BEM-II、VSOPというBDD/ZDD処理インタプリタは公開済み
- ERATO期間内に整備してオープンソースとして公開したい。
  - 現在のコードを仕様書として、再コーディングを外注するかも。
  - ドキュメントの整備、一定期間のサポート体制
  - Rubyのようなスクリプト言語版も整備したい。
  - 他のBDDパッケージとの性能比較もたぶんやった方がよい。

北大版パッケージの主な仕様

- Linux, gnu C コンパイラを想定
  - コア部分は 性能重視でC言語で記述
  - 周りをC++のインターフェースで包んである。
    (bddcopy, bddfree のメモリ管理を C++ で自動化)
  - さらに拡張ライブラリはC++で記述
- x86互換32ビット機向けに設計しているが、
  64ビット機(long long int)にも対応可能
  - 単純に拡張するとノード当たりのメモリが2倍になるが、
    48ビットと16ビットに区切って詰め込んで、メモリ増加を抑制
  - コンパイルオプションの指定で64ビット向けにコンパイル
    (一部のmakefileを手動で書き直す必要あり)
- Cygwin環境でも動作
  - WindowsのノートPCでデモ可能

北大版BDDパッケージの構成

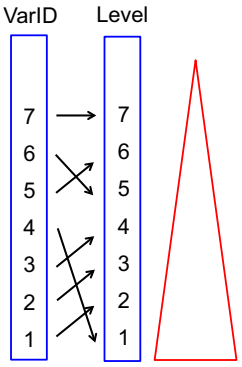
- TARファイルとして固めてある
  - 解凍するとSAPPOROBDD2010xxというディレクトリができる。
  - その下に、app, include, lib, man, src というサブディレクトリ。
  - src の下に移動して ./INSTALL を実行すると、BDDパッケージが
    コンパイルされて、lib の下に BDD.a というライブラリができる。
  - 基本的に BDD.a をリンクすればBDDのC++ライブラリが使える
  - 使い方は man の下にある。(現状、日本語版のみ)
- BEM-II と VSOP のインストール
  - 上記のライブラリコンパイルの後、app の下に行って、
    BemII または VSOP の下で、make を実行すと、
    プログラム bemII または vsop がコンパイルされる。
  - bemII.help および vsop.help を実行可能にしておく。
    (パスを通して、パーミッションを実行可能に)

北大版BDDパッケージのドキュメント

- bddc.pdf
  - BDDパッケージのコア部分のデータ構造と
    外部公開の関数コールの説明
- BDD+.pdf
  - C++版のクラス構成と、メソッド関数の説明
- bemII.pdf
  - BEM-II の起動法と主なコマンドの説明
- vsop.pdf
  - VSOP の起動法と主なコマンドの説明

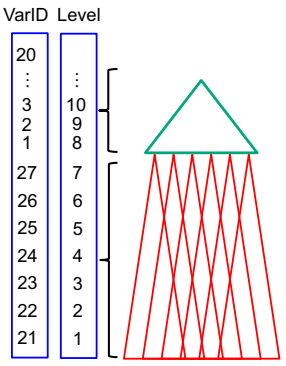
(補足)変数番号 VarID と Level

- BDD節点は下位から順に1,2,3,…の
  Levelを持つ。
- 変数番号はVarIDで識別される。
  それぞれにLevelを持つ。
  - VarIDを触らずに後からLevelだけを変えられるようにするため、間接参照する形になっている。
- bddnewvar() を呼び出す度に、
  1つ大きなVarIDが振り出される。
- bddnewvaroflev(level)でlevelを指定してVarIDを振り出せる。
- bddvaroflev(), bddlevofvar() で相互参照換できるようになっている。



(補足)BDDVで用いる特殊変数

- BDDV_Init()で初期化した場合には、
  BDDのベクトルを扱うための特殊変数が最初に20個確保されるので、
  BDD_NewVar() のVarIDは 21番からスタートする。



********** BDD パッケージ（1.21 版）ドキュメント **********

このパッケージは BDD の基本操作を行うＣ言語の関数ライブラリである。

・本プログラムは、32 ビットまたは 64 ビットの計算機で動作する。（コンパイル時に、オプション B_64 を指定すると 64 ビットモードとなる）

・各操作はＣ言語の関数呼び出しにより実行される。
関数等の宣言部は、bddc.h にあるので、include すること。
プログラム本体は bddc.c にある。

・入力変数番号(VarID)は 1 から始まる unsigned int 型（bddvar 型）の整数で識別する（0 は定数を表す）。VarID の最大値は bddvarmax で参照される。デフォルトは 65535(16 ビット)。

・各 VarID ごとに BDD での上下の順位(level)の情報を保持している。Level もまた 1 から始まる bddvar 型の整数で識別する。大きい数値ほど上位の変数を表す（BDD の根に近く、先に展開される）。VarID を何も指定せずに生成した場合は VarID と同じ値の level を持つ。

・論理演算結果の BDD は、32 ビット（または 64 ビット）の unsigned int（bddp 型）のインデックスで返される。BDD は論理関数に対して一意であり、インデックスの値も BDD に対して一意である。したがって、２つの論理演算結果が等価であるかどうかは、演算結果のインデックスの値が同じかどうかを比較することで行える。

```
bddp f, g
......
if(f == g) 一致
if(f != g) 不一致
```

ただしインデックスの大小比較（f ＞ g）は意味を持たない。

・BDD のインデックスのビット構造は以下の通りである
ビット幅 32 の場合
ABBBBBBBBBBBBBBBBBBBBBBBBBBBBC

ビット幅 64 の場合（下位 40 ビットを使用）
OOOOOOOOOOOOOOOOOOOOOOOOAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBC
A: 節点 ID か定数値かを区別するフラグ。0:節点 ID　1:定数値
B:　（A=0 の場合）節点番号を表す。
　　（A=1 の場合）BC を合わせて定数値を表す。
　　　　通常の論理関数の場合、最下位ビット C 以外は常に 0。
C: 否定枝を表すフラグ。定数値の場合は奇数偶数を表す。

・BDD の節点情報を格納する記憶領域のデータ構造を以下に示す。
ビット幅 32 の場合
Node.f0_32　ABBBBBBBBBBBBBBBBBBBBBBBBBBBBC 0 枝の BDD インデックス
Node.f1_32　ABBBBBBBBBBBBBBBBBBBBBBBBBBBBC 1 枝の BDD インデックス
Node.nx_32　OOBBBBBBBBBBBBBBBBBBBBBBBBBBBB N:次の節点番号（記憶管理用）
Node.varrfc RRRRRRRRRRRRRRRRVVVVVVVVVVVVVVVV R:参照カウンタ V:変数番号

ビット幅 64 の場合
Node.f0_32　BBBBBBBBBBBBBBBBBBBBBBBBBBBBBC 0 枝の BDD インデックス
Node.f1_32　BBBBBBBBBBBBBBBBBBBBBBBBBBBBBC 1 枝の BDD インデックス
Node.nx_32　BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB N:次の節点番号（記憶管理用）
Node.varrfc RRRRRRRRRRRRRRRRVVVVVVVVVVVVVVVV R:参照カウンタ V:変数番号
Node.f0_h8　ABBBBBBB　0 枝の BDD インデックス（上位 8 ビット）
Node.f1_h8　ABBBBBBB　1 枝の BDD インデックス（上位 8 ビット）
Node.nx_h8　OOBBBBBB　N:次の節点番号（記憶管理用）（上位 8 ビット）

・本プログラムでは、各 BDD の参照回数を記憶するカウンタ（参照カウンタ）を用いて記憶管理を行っている。BDD をコピーする際には、インデックスを直接代入せず、必ず bddcopy() を使用する。また、不要になった変数は、bddfree() にて解放することにより、記憶の再利用が行われる。

・論理演算中に記憶あふれが発生した場合は、その演算を行う前の状態に戻し、bddnull を返す。それ以外のエラーが発生した場合は、エラーメッセージを出力した後、異常終了する。なお、関数の引数に bddnull を与えた場合には、基本的に何もしないで bddnull を返す。

・本プログラムでは、組合せ集合を表す Zero-suppressed BDD (ZBDD)の処理も行う。ZBDD と BDD の節点は内部で区別されている。ZBDD 向けの演算の引数に BDD の節点を与えた場合（またはその逆も）エラーを検出し異常終了する。ちなみに内部での区別の仕方であるが，BDD/ZBDD では，0 枝側は否定枝にはならないという性質があるため，基本的に f0_32 の最下位ビットは 0 になっているはずである．そこで，本来 0 であるべき最下位ビットが 1 になっていたら ZBDD 節点であることを示している。

***** 定数マクロ *****

```
bddvarmax   入力変数番号の最大値（通常 65535）
bddfalse    恒偽関数を指す BDD インデックス値（0x80000000）
bddtrue     恒真関数を指す BDD インデックス値（0x80000001）
bddnull     エラーを意味する BDD インデックス値（通常 0x7FFFFFFF）
bddempty    ZBDD の空集合を表す BDD インデックス値（= bddfalse）
bddsingle   ZBDD の単位元集合を表す BDD インデックス値（= bddtrue）
```

***** 関数 *****

---------- [1] 初期設定・入力変数番号設定 ----------

```
extern int bddinit(bddp initsize, bddp limitsize)
```

処理系を初期化し、メモリの確保を行う。プログラムの最初に必ず実行しなければならない。initsize で、最初にメモリを確保する BDD 節点数を指定する。以後、演算中にメモリを使い切った場合は、自動的にメモリの再確保が行われる。再確保毎に節点数は 4 倍に拡大される。拡大の上限は、limitsize によって指定できる。使用節点数が limitsize に達したときは、メモリの再確保はそれ以上行われず、ガベジコレクションが起動され、bddfree()により解放された空き節点が回収される。initsize は、最低 256 より大きく、limitsize を越えてはならない。limitsize は、最低 256 より大きく、上限は計算機の実メモリ量に依存する。(1 節点当たり約 25 バイト必要とする。)
bddinit()による初期化が正常に行われた場合には、関数の値として 0 を返し、メモリ確保に失敗した場合 1 を返す。bddinit()を複数回実行すると、前回の内容がクリアされ、再度初期化される。

```
extern bddvar bddnewvar(void)
```

新しい入力変数を 1 つ生成し、その変数番号(VarID)を返す。VarID は 1 から始まる整数で、bddnewvar()または bddnewvaroflev()を 1 回実行するごとに 1 ずつ大きな値が返る。生成した変数の BDD 展開順位(level)は、VarID と同じ値となる。変数の個数が最大値 bddvarmax を超えるとエラーを出力して異常終了する。

```
extern bddvar bddnewvaroflev(bddvar lev)
```

新しい入力変数を 1 つ生成し、その変数番号(VarID)を返す。VarID は 1 から始まる整数で、bddnewvar()または bddnewvaroflev を 1 回実行するごとに] 1 ずつ大きな値が返る。生成した変数の BDD 展開順位(level)は、引数 lev で指定した値となる。実行時に順位 lev の変数がすでに存在していた場合は、lev 以上の変数を 1 つずつ上にずらして（level を 1 ずつ増加させ)、空いたところに新しい変数を挿入する。引数 lev は 1 以上かつ「関数実行直前の変数の個数＋1」以下でなければならない。そうでなければエラーを出力して異常終了する。変数の個数が最大値 bddvarmax を超えるとエラーを出力して異常終了する。

```
extern bddvar bddlevofvar(bddvar v)
```

引数 v で指定した変数番号(VarID) の BDD 展開順位(level)を返す。引数 v は 1 以上かつ「現在の変数の個数」以下でなければならない。そうでなければエラーを出力して異常終了する。

```
extern bddvar bddvaroflev(bddvar lev)
```

引数 lev で指定した BDD 展開順位(level)を持つ変数番号(VarID)を返す。
引数 lev は 1 以上かつ「現在の変数の個数」以下でなければならない。
そうでなければエラーを出力して異常終了する。

```
extern bddvar bddvarused(void)
```

現在の入力変数の個数を返す。

---------- [2] 基本的な論理演算 ----------

extern bddp　bddprime(bddvar v)

変数番号 v の入力変数に関する単項関数を表す BDD を作り、それを指すインデックスを返す。すでに同じ BDD が存在していれば共有し、参照カウンタの値を１増やす。引数 v は１以上かつ「現在の変数の個数」以下でなければならない。実行中に記憶あふれを起こした場合は、bddnull を返す。不当な引数を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD ではない通常の BDD を生成する。ZBDD 向け処理には利用できない。

extern bddvar bddtop(bddp f)

f が指す BDD の最上位の節点の変数番号(VarID)を返す。返すのは BDD 展開順位の値(level)ではなくその値を持つ変数の VarID であることに注意。f の参照カウンタの値は変化しない。f が定数関数の場合は 0 を返す。引数に bddnull を与えた場合は 0 を返す。不当な引数（BDD を正しく指していない等）を与えた場合はエラーを出力し異常終了する。この演算は BDD, ZBDD 共に利用可能。

extern bddp　bddcopy(bddp f)

f が指す BDD をコピーする。すなわち、参照カウンタの値を１増やし、f そのものを返す。bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は BDD,ZBDD 共に利用可能。

extern bddp　bddnot(bddp f)

f の否定を表す BDD を作り、それを指すインデックスを返す。「否定枝」を使用しているため、節点数は増加せず、f の参照カウンタの値を１増やすだけで、定数時間で結果を返す。bddnul を与えた場合は bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddand(bddp f, bddp g)

f と g の論理積を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddor(bddp f, bddp g)

f と g の論理和を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddxor(bddp f, bddp g)

f と g の排他的論理和を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddnand(bddp f, bddp g)

f と g の論理積の否定を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddnor(bddp f, bddp g)

f と g の論理和の否定を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddxnor(bddp f, bddp g)

f と g の排他的論理和の否定を表す BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラーを出力し異常終了する。

extern bddp　bddat0(bddp f, bddvar v)

f が指す BDD に対して、変数番号 v の入力変数に 0 を代入したときの BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f が ZBDD を指していた場合はおかしな計算をする可能性がある。

extern bddp　bddat1(bddp f, bddvar v)

f が指す BDD に対して、変数番号 v の入力変数に 1 を代入したときの BDD を作り、それを指すインデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この演算は ZBDD では定義されていないため、f が ZBDD を指していた場合はおかしな計算をする可能性がある。

---------- [3] 記憶管理・表示 ----------

extern void　bddfree(bddp f)

f が指す BDD がもはや不要であることを宣言する。すなわち、参照カウンタの値を１減らす。定数関数の場合は何もしない。f が指していた BDD は、ガベジコレクションが起動されるまでは、回収されずに残っている。
引数に bddnull を与えた場合は、何もしない。この演算は BDD,ZBDD 共に利用可能。

extern bddp　bddused(void)

現在使用中の総節点数を返す。bddfree() によって解放された節点も、回収されるまでは使用中として数えるため、正確な節点数を知るには、直前に bddgc()を実行（ガベジコレクション起動）する必要がある。

extern int　bddgc(void)

強制的にガベジコレクション（不要な節点の回収）を行う。bddgc()を陽に起動しなくても、記憶が足りなくなった場合には自動的に起動される。
ガベジコレクションで空き節点が回収された場合は 0 を返し、空き節点が１個も見つからなかった場合は 1 を返す。

extern bddp　bddsize(bddp f)

f が指すの BDD の節点数を返す。参照カウンタの値は変化しない。
引数に bddnull を与えた場合は、0 を返す。不当な引数（BDD を正しく

指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は BDD,ZBDD 共に利用可能。

```
extern bddp   bddvsize(bddp *p, int lim)
```

bddp 型の配列 p[]（配列長の上限 lim）により与えられた複数の BDD の節点数を返す。
複数の BDD に共通に含まれる節点は重複して数えない。参照カウンタの値は変化しない。
配列の要素として bddnull が出現したら、その直前で配列が終了しているとする。
bddnull が出現しなければ配列長は lim までとする。配列の記憶領域はあらかじめ
確保されているものとする。不当な引数を与えた場合はエラーを出力し異常終了する。
この演算は BDD,ZBDD 共に利用可能。

```
extern void   bddexport(FILE *strm, bddp *p, int lim)
```

bddp 型の配列 p[]（配列長の上限 lim）により与えられた複数の BDD の構造を、
strm で指定するファイルに出力する。配列の要素として bddnull が出現したら、
その直前で配列が終了しているとする。bddnull が出現しなければ配列長は
lim までとする。配列の記憶領域はあらかじめ確保されているものとする。
不当な引数を与えた場合はエラーを出力し異常終了する。
この演算は BDD,ZBDD 共に利用可能。

```
extern int   bddimport(FILE *strm, bddp *p, int lim)
```

strm で指定するファイルから BDD の構造を読み込み、bddp 型の配列 p[](配列長の上限 lim)
に格納する。読み込んだ BDD 配列の要素数の１つ先の要素に bddnull を書き込む。
ただし、ファイルに書かれているデータの配列長が lim より大きいときは lim まで
しか読まない。配列の記憶領域はあらかじめ確保されているものとする。
ファイルに文法誤りが合った場合等、異常終了時は 1 を返す。正常時は 0 を返す。
この演算は BDD でのみ正しく動作する。

```
extern int   bddimportz(FILE *strm, bddp *p, int lim)
```

strm で指定するファイルから ZBDD の構造を読み込み、bddp 型の配列 p[]（配列長の上限
lim）に格納する。読み込んだ ZBDD 配列の要素数の１つ先の要素に bddnull を書き込む。
ただし、ファイルに書かれているデータの配列長が lim より大きいときは lim まで
しか読まない。配列の記憶領域はあらかじめ確保されているものとする。

ファイルに文法誤りが合った場合等、異常終了時は 1 を返す。正常時は 0 を返す。
この演算は ZBDD でのみ正しく動作する。

```
extern void   bddgraph(bddp f)
```

f が指す BDD のグラフ構造を X-Window に描画する。引数に bddnull を与
えた場合は、何も表示しない。不当な引数（BDD を正しく指していな
い等）を与えた場合は、エラーを出力し異常終了する。この演算は BDD,ZBDD
共に利用可能。

```
extern void   bddgraph0(bddp f)
```

f が指す BDD のグラフ構造を X-Window に描画する。bddgraph()とほとんど
同じだが、否定枝を使わない場合のグラフ構造を描画する。引数に bddnull
を与えた場合は、何も表示しない。不当な引数（BDD を正しく指して
いない等）を与えた場合は、エラーを出力し異常終了する。
この演算は ZBDD では正しく表示されない。

```
extern void   bddvgraph(bddp *p, int lim)
```

bddp 型の配列 p[]（配列長 n）により与えられた複数の BDD ののグラフ構造
を X-Window に描画する。配列の要素として bddnull が出現したら、その直前で
配列が終了しているとする。bddnull が出現しなければ配列長は lim までとする。
配列の記憶領域はあらかじめ確保されているものとする。不当な引数
（BDD を正しく指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は BDD,ZBDD 共に利用可能。

```
extern void   bddvgraph0(bddp *p, int n)
```

bddp 型の配列 p[]（配列長 n）により与えられた複数の BDD ののグラフ構造
を X-Window に描画する。bddvgraph()とほとんど同じだが、否定枝を使わ
ない場合のグラフ構造を描画する。配列の要素として bddnull が出現したら、
その直前で配列が終了しているとする。bddnull が出現しなければ配列長は
lim までとする。配列の記憶領域はあらかじめ確保されているものとする。
不当な引数（BDD を正しく指していない等）を与えた場合は、
エラーを出力し異常終了する。この演算は ZBDD では正しく表示されない。

---------- [4] その他の論理演算 ----------

```
extern bddp   bddlshift(bddp f, bddvar shift)
```

f が指す BDD について、関係する全ての入力変数を、展開順位(level)が
shift ずつ大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて
BDD を複製し、それを指すインデックスを返す。実行結果において未定義の入力
変数が必要になるような shift を与えてはならない。必要な入力変数は
あらかじめ宣言しておくこと。計算結果と同じ BDD がすでに存在していれば
共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした
場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。
shift に負の値を指定することはできない。不当な引数（BDD を正しく指し
ていない等）を与えた場合は、エラーを出力し異常終了する。この演算は
BDD,ZBDD 共に利用可能。

```
extern bddp   bddrshift(bddp f, bddvar shift)
```

f が指す BDD について、関係する全ての入力変数を、展開順位(level)が
shift ずつ大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて
BDD を複製し、それを指すインデックスを返す。実行結果において未定義の入力
変数が必要になるような shift を与えてはならない。したがって、f に関係
しない入力変数が下位レベルに用意されていなければならない。計算結果と
同じ BDD がすでに存在していれば共有し、参照カウンタの値を１増やす。
実行中に記憶あふれを起こした場合は、bddnull を返す。引数に bddnull を
与えた場合は、bddnull を返す。shift に負の値を指定することはできない。
不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し
異常終了する。この演算は BDD,ZBDD 共に利用可能。

```
extern bddp   bddsupport(bddp f)
```

f が指す BDD に関係する入力変数（変数の値が 0 か 1 かで f の結果が異なる
ような変数）の集合を取り出す。演算結果は、関係する入力変数の論理和
（例: a + b + d）を表す BDD を生成し、それを指すインデックスを返す。
（演算結果の BDD の 0 枝を順にたどっていくと、求める変数が得られる）

f が定数の場合は bddfalse を返す。不当な引数（BDD を正しく指して
いない等）を与えた場合は、エラーを出力し異常終了する。この演算は
BDD,ZBDD 共に利用可能。f が ZBDD の場合は演算結果は ZBDD の集合和の
形式となる。

```
extern bddp   bdduniv(bddp f, bddp g)
```

全称作用演算(universal quantification)。g で指定した入力変数の部分集合
に 0,1 の定数を代入したときに、どのような 0,1 の組合せを代入しても常に f=1
となる場合には 1 を返し、それ以外は 0 を返すような論理関数の BDD を作り、
それを指すインデックスを返す。入力変数の部分集合の与え方は、bddsupport()
の場合と同様で、変数の論理和の形式とする。(g が指す BDD の 0 枝を順に
たどっていくと、求める変数が得られる。) 計算結果と同じ BDD がすでに存在
していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした
場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。
不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し
異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を
指していた場合はエラーを出力し異常終了する。

```
extern bddp   bddexist(bddp f, bddp g)
```

存在作用演算(existential quantification)。g で指定した入力変数の部分集合
に 0,1 の定数を代入したときに、どのような 0,1 の組合せを代入しても常に f=0
となる場合には 0 を返し、それ以外は 1 を返すような論理関数の BDD を作り、
それを指すインデックスを返す。入力変数の部分集合の与え方は、bddsupport()
の場合と同様で、変数の論理和の形式とする。(g が指す BDD の 0 枝を順に
たどっていくと、求める変数が得られる。) 計算結果と同じ BDD がすでに存在
していれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起こした
場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。
不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し
異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を
指していた場合はエラーを出力し異常終了する。

```
extern bddp   bddcofactor(bddp f, bddp g)
```

g = 0 のときを don't care として f を簡単化した BDD を作り、それを指す

インデックスを返す。計算結果と同じ BDD がすでに存在していれば共有し、
参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、bddnul
を返す。引数に bddnull を与えた場合は、bddnull を返す。不当な引数（BDD を
正しく指していない等）を与えた場合は、エラーを出力し異常終了する。この
演算は ZBDD では定義されていないため、f,g が ZBDD を指していた場合はエラー
を出力し異常終了する。

```
extern bddp   bddimply(bddp f, bddp g)
```

f -> g （f が真ならば g は真）が恒に成り立つかを調べ、恒に成り立つなら
bddtrue を返し、１つでも反例があれば bddfalse を返す。実行中に節点数は増加
しないので効率が良い。引数に bddnull を与えた場合は、bddfalse を返す。
不当な引数（BDD を正しく指していない等）を与えた場合は、エラーを出力し
異常終了する。この演算は ZBDD では定義されていないため、f,g が ZBDD を
指していた場合はエラーを出力し異常終了する。

```
extern void   bddwcache(unsigned char op, bddp f, bddp g, bddp h)
```

演算結果キャッシュへの登録を行う。op は演算の種類を表す。(f op g) =
h という演算結果を登録する。f, g, h は、bddp 型のデータを与える。
引数エラーチェックは行っていないので注意。この演算は BDD,ZBDD 共に
利用可能である。なお、1.00 版では、op = 0～9 は、BDD 処理系内部の
演算用に、op = 10～19 は、ZBDD 関係の演算用に使用されており、
20 以上の番号が未使用である。複数のアプリケーションで番号が衝突
しないように注意が必要。

```
extern bddp   bddrcache(unsigned char op, bddp f, bddp g)
```

演算結果キャッシュを参照する。過去に同じ演算が登録されていれば、
演算結果の BDD へのインデックスを返し、見つからなければ bddnull
を返す。ただし、値を返すだけで、参照カウンタの処理は行わないため、
呼び出し側で bddcopy()を実行する必要がある。引数エラーのチェックは
行っていないので注意。

---------- [5] ZBDD 用の組合せ集合演算 ----------

```
extern bddp   bddoffset(bddp f, bddvar v)
```

f が指す ZBDD において、入力変数番号 v のアイテムを含まない組合せ要素
を集めた部分集合を表す ZBDD を作り、それを指すインデックスを返す。
計算結果と同じ ZBDD がすでに存在していれば共有し、参照カウンタの値を
１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。
引数に bddnull を与えた場合は、bddnull を返す。不当な引数（ZBDD を正
しく指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は ZBDD 専用であるため、f が通常の BDD を指していた場合は
エラーを出力し異常終了する。

```
extern bddp   bddonset(bddp f, bddvar v)
```

f が指す ZBDD において、入力変数番号 v のアイテムを含む組合せ要素
を集めた部分集合を表す ZBDD を作り、それを指すインデックスを返す。
計算結果と同じ ZBDD がすでに存在していれば共有し、参照カウンタの値を
１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。
引数に bddnull を与えた場合は、bddnull を返す。不当な引数（ZBDD を正
しく指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は ZBDD 専用であるため、f が通常の BDD を指していた場合は
エラーを出力し異常終了する。

```
extern bddp   bddonset0(bddp f, bddvar v)
```

bddonset() とほぼ同じであるが、抽出した部分集合の各要素から
入力変数番号 v のアイテムが取り除かれている。
bddchange(bddonset(f, rank), rank) と等価である。
v が f の最上位の変数番号であれば、1-枝の指す BDD をそのまま返す。
計算結果と同じ ZBDD がすでに存在していれば共有し、参照カウンタの値を
１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。
引数に bddnull を与えた場合は、bddnull を返す。不当な引数（ZBDD を正
しく指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は ZBDD 専用であるため、f が通常の BDD を指していた場合は
エラーを出力し異常終了する。

```
extern bddp   bddchange(bddp f, bddvar v)
```

f が指す ZBDD に含まれる全ての組合せ要素について、入力変数番号 v の
アイテムの有無を反転させた組合せ集合を表す ZBDD を作り、それを指す
インデックスを返す。計算結果と同じ ZBDD がすでに存在していれば共有し、
参照カウンタの値を１増やす。実行中に記憶あふれを起こした場合は、
bddnull を返す。引数に bddnull を与えた場合は、bddnull を返す。
不当な引数（ZBDD を正しく指していない等）を与えた場合は、エラーを
出力し異常終了する。この演算は ZBDD 専用であるため、f が通常の BDD を
指していた場合はエラーを出力し異常終了する。

```
extern bddp   bddintersec(bddp f, bddp g)
```

f と g の積集合を表す ZBDD を作り、それを指すインデックスを返す。
計算結果と同じ ZBDD がすでに存在していれば共有し、参照カウンタの値を
１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に
bddnull を与えた場合は、bddnull を返す。不当な引数（ZBDD を正しく指
していない等）を与えた場合は、エラーを出力し異常終了する。この演算は
ZBDD 専用のため、f,g が通常の BDD を指していた場合はエラーを出力し
異常終了する。

```
extern bddp   bddunion(bddp f, bddp g)
```

f と g の和集合を表す ZBDD を作り、それを指すインデックスを返す。
計算結果と同じ ZBDD がすでに存在していれば共有し、参照カウンタの値を
１増やす。実行中に記憶あふれを起こした場合は、bddnull を返す。引数に
bddnull を与えた場合は、bddnull を返す。不当な引数（ZBDD を正しく指
していない等）を与えた場合は、エラーを出力し異常終了する。この演算は
ZBDD 専用のため、f,g が通常の BDD を指していた場合はエラーを出力し
異常終了する。

```
extern bddp   bddsubtract(bddp f, bddp g)
```

f と g の差集合（f に含まれ g に含まれていない要素）を表す ZBDD を作り、
それを指すインデックスを返す。計算結果と同じ ZBDD がすでに存在して
いれば共有し、参照カウンタの値を１増やす。実行中に記憶あふれを起

こした場合は、bddnull を返す。引数に bddnull を与えた場合は、bddnull
を返す。不当な引数（ZBDD を正しく指していない等）を与えた場合は、
エラーを出力し異常終了する。この演算は ZBDD 専用のため、f,g が通常の
BDD を指していた場合はエラーを出力し異常終了する。

```
extern bddp   bddcard(bddp f)
```

f が指す ZBDD に含まれる要素数を返す。引数に bddnull を与えた場合は、
0 を返す。参照カウンタの値は変化しない。不当な引数（ZBDD を正しく
指していない等）を与えた場合は、エラーを出力し異常終了する。
この演算は ZBDD 専用のため、f が通常の BDD を指していた場合は
エラーを出力し異常終了する。

```
extern bddp   bddlit(bddp f)
```

f が指す ZBDD に含まれる組合せに出現するアイテム数の総和を返す。
引数に bddnull を与えた場合は、0 を返す。参照カウンタの値は
変化しない。不当な引数（ZBDD を正しく指していない等）を与えた
場合は、エラーを出力し異常終了する。この演算は ZBDD 専用のため、
f が通常の BDD を指していた場合はエラーを出力し異常終了する。

```
extern bddp   bddlcma(char *fname, int th)
```

fname で指定する名前のファイルから FIMI ベンチマークフォーマットのトランザクション
データベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する頻出アイ
テム集合を表す ZBDD を生成し、それを指すインデックスを返す。記憶あふれの場合が
bddnull を返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

```
extern bddp   bddlcmc(char *fname, int th)
```

fname で指定する名前のファイルから FIMI ベンチマークフォーマットのトランザクション
データベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する飽和頻出
アイテム集合を表す ZBDD を生成し、それを指すインデックスを返す。記憶あふれの場合
が bddnull を返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

```
extern bddp   bddlcmm(char *fname, int th)
```

fnameで指定する名前のファイルからFIMIベンチマークフォーマットのトランザクションデータベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する極大頻出アイテム集合を表す ZBDD を生成し、それを指すインデックスを返す。記憶あふれの場合が bddnull を返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

********** C++版 BDD パッケージ(SAPPORO-1.31)　ドキュメント **********
著者: 湊　真一　北海道大学　大学院情報科学研究科
最新更新日: 2011.3.21
****

パッケージの概要
・このパッケージは BDD の基本操作を行う C++のクラスライブラリである。
　本プログラムは、32 ビットまたは 64 ビットの計算機で動作する。
　（コンパイル時に、オプション B_64 を指定すると 64 ビットモードとなる）
　各操作は C++のメソッド呼び出しにより実行される。

・入力変数番号(通称 VarID)は 1 から始まる int 型の整数で識別する（0 は定数を表す)。
　負の変数番号は用いない。VarID の最大値は定数 BDD_MaxVar で与えられる。
　デフォルトは 65535(16 ビット)。

・各 VarID ごとに BDD での上下の順位(通称 level)の情報を保持している。
　level もまた 1 から始まる int 型の整数で識別する。大きい数値ほど
　上位の変数を表す（BDD の根に近く、先に展開される)。
　VarID を何も指定せずに生成した場合は VarID と同じ値の level を持つ。

・論理演算結果の BDD は、32 ビット（または 64 ビット）の unsigned int
　(bddword という名前の型に typedef されている）のインデックスで返される。
　BDD は論理関数に対して一意であり、インデックスの値も BDD に対して
　一意である。したがって、２つの論理演算結果が等価であるかどうかは、
　演算結果のインデックスの値が同じかどうかを比較することで行える。

・BDD 節点テーブルの最大サイズは、BDD_Init() の２つの引数で指定する。
　BDD_Init を省略した場合の default は、初期値 256、最大値 1,024 に
　設定されている。計算中に記憶あふれを起こした場合は、計算を中断して、
　null オブジェクト BDD(-1) を返す。

・提供するクラスとその依存関係:
```
BDD                 BDD で表現された個々の論理関数を指すクラス
 |-- BDDV           BDD の配列（論理関数の配列）を表すクラス
 |    |-- Btol      ２値入力整数値出力の論理関数を表すクラス
 |
 |-- BDDDG          BDD を単純直交分解した結果を表すクラス
 |- ZBDD            ゼロサプレス型 BDD で表現された組合せ集合を指すクラス
     |-- ZBDDV      ZBDD の配列（組合せ集合の配列）を表すクラス
     |    |-- Ctol  整数値組合せ集合（整係数ユネイト論理式）を表すクラス
     |
     |-- SOP        正負のリテラルからなる積和形論理式を表現するクラス
     |    |-- SOPV  SOP の配列（積和形論理式の配列）を表すクラス
     |
     |-- PiDD       順列集合を表現するクラス
     |
     |-- ZBDDDG     ZBDD を単純直交分解した結果を表すクラス
```

・BDD クラスの使用例
```
int x = BDD_NewVar();
int y = BDD_NewVar();
BDD f1 = BDDvar(x);
BDD f2 = BDDvar(y);
BDD f3 = ~ f1 & f2;
BDD f4 = (~f1 ^ f3) | f2;
f3.Print();
f4.Print();
```

****
クラス名: BDD --- BDD で表現された個々の論理関数を指すクラス
****
ヘッダーファイル名: "BDD.h"
ソースファイル名: BDD.cc
内部から呼び出しているクラス: （無し）

----------関連する定数値----------
extern const bddword BDD_MaxNode
１ワードで区別できる節点数の最大値。32 ビットマシンでは 2 の 30 乗に、
64 ビットマシンでは２の 38 乗にセットされているが、メモリ容量の限界が
あるので、実際にはもっと少ない個数しか扱うことはできない。
現実の最大節点数は、BDD_Init()の引数で指定する。

extern const int BDD_MaxVar
入力変数番号の最大値。通常 65535。

----------関連する外部関数----------
void BDD_Init(bddword init, bddword limit)
処理系を初期化しメモリの確保を行う。bddword は unsigned int の別名である。
引数 init で、最初にメモリを確保する BDD 節点数を指定する。以後、演算中
にメモリを使い切った場合は、自動的にメモリの再確保が行われる。再確保毎に
節点数は 4 倍に拡大される。拡大の上限は、引数 limit によって指定できる。
使用節点数が limit に達したときは、メモリの再確保はそれ以上行われず、
ガベジコレクションが起動され、空き節点が自動的に回収される。init は、
256 以上でなければならず 、limit を越えてはならない。limit は、256 以上でなければ
ならず、上限は計算機のメモリ容量に依存する。(1 節点当たり約 25 バイト必要とする。)
BDD_Init()による初期化が正常に行われた場合には、関数の値として 0 を返し、
メモリ確保に失敗した場合 1 を返す。BDD_Init()を複数回実行すると、前回の
内容がクリアされ、再度初期化される。BDD_Init()を一度も実行せずに演算を開始した
場合は、init=256, limit=1024 が仮定される。

extern int BDD_NewVar(void)
新しい入力変数を１つ生成し、その変数番号(通称 VarID)を返す。VarID は 1 から
始まる整数で、BDD_NewVar()または BDD_NewVarOfLev()を 1 回実行するごとに
1 ずつ大きな値が返る。生成した変数の BDD 展開順位(通称 level)は、VarID と
同様に下位から順番に割り当てられる。変数の個数が最大値 BDD_MaxVar を
超えるとエラーを出力して異常終了する。
なお、最初に BDDV_Init()で初期化した場合（BDDV クラスを扱う場合）には、
最初にシステム用に変数が使われるので、VarID は (BDDV_SysVarTop + 1)から
開始し、順に 1 ずつ大きな値となる。

extern int BDD_NewVarOfLev(int lev)
新しい入力変数を１つ生成し、その変数番号(通称 VarID)を返す。VarID は 1 から
始まる整数で、BDD_NewVar()または BDD_NewVarOfLev()を 1 回実行するごとに
1 ずつ大きな値が返る。生成した変数の BDD 展開順位(通称 level)は、引数 lev で
指定した値となる。実行時に順位 lev の変数がすでに存在していた場合は、
lev 以上の変数を１つずつ上にずらして（level を１ずつ増加させて)、空いた
ところに新しい変数を挿入する。引数 lev は 1 以上かつ「関数実行直前の変数の
個数＋１」以下でなければならない。そうでなければエラーを出力して異常終了する。
変数の個数が最大値 BDD_MaxVar を超えるとエラーを出力して異常終了する。

extern int BDD_LevOfVar(int v)
引数 v で指定した変数番号(通称 VarID) の BDD 展開順位(通称 level)を返す。
引数 v は 1 以上かつ「現在の変数の個数」以下でなければならない。
そうでなければエラーを出力して異常終了する。

extern int BDD_VarOfLev(int lev)
引数 lev で指定した BDD 展開順位(通称 level)を持つ変数番号(通称 VarID)を返す。
引数 lev は 1 以上かつ「現在の変数の個数」以下でなければならない。
そうでなければエラーを出力して異常終了する。

extern int BDD_VarUsed(void)
現在までに宣言した入力変数の個数を返す。

extern int BDD_TopLev(void)
現在までに宣言した入力変数の最上位変数の順位(Level)を返す。
最初に BDDV_Init()で初期化した場合（BDDV クラスを扱う場合）には、
BDD_VarUsed() + BDDV_SysVarTop に等しい。
BDD_Init()で初期化した場合は、BDD_VarUsed() と等しい。

extern bddword BDD_Used(void)
現在使用中の総節点数を返す。使用済みで再利用可能な節点も、実際に
回収されるまでは使用中として数えるため、正確な節点数を知るには、
直前に BDD_GC()を実行（ガベジコレクション起動）する必要がある。

extern void BDD_GC(void)
強制的にガベジコレクション（不要な節点の回収）を行う。BDD_GC()を陽に
起動しなくても、記憶が足りなくなった場合には自動的に起動される。
ガベジコレクションで空き節点が回収された場合は 0 を返し、空き節点が
１個も見つからなかった場合は 1 を返す。

extern bddword BDD_CacheInt(unsigned char op, bddword f, bddword g)
f と g の演算結果が数値のとき、演算結果をキャッシュから参照する。
引数 op は演算の種類を表す番号で、20 以上の値を入れる。演算結果が
登録されている場合はその数値を返し、見つからなかった場合は、
BDD_MaxNode よりも大きな値を返す。f, g が BDD 型の演算の場合は、
GetID()で bddword 型に変換して与える。

extern BDD BDD_CacheBDD(unsigned char op, bddword f, bddword g)
f と g の演算結果が BDD 型のとき、演算結果をキャッシュから参照する。
op は演算の種類を表す番号で、20 以上の値を入れる。演算結果が登録されて
いる場合はその BDD を返し、見つからなかった場合は、null を表すオブジェ
クトを返す。f, g が BDD 型の演算の場合は、GetID()で bddword 型に変換して与える。

extern void BDD_CacheEnt(unsigned char op, bddword f, bddword g, bddword h)
f と g の演算結果 h をキャッシュに登録する。op は演算の種類を表す番
号で、20 以上の値を入れる。被演算子や演算結果が数値のときは、そのま
ま与える。BDD 型の演算の場合は、GetID()で bddword 型に変換して与える。

extern BDD BDDvar(int var)
入力変数番号 var の変数そのもの（恒等関数）を表す BDD オブジェクトを
生成し、それを返す。記憶あふれの場合は、null を表すオブジェクトを返す。

extern BDD operator&(const BDD& f, const BDD& g)
f と g の論理積を表す BDD オブジェクトを生成し、それを返す。記憶あふれの
場合は、null を表すオブジェクトを返す。引数に null を与えた場合には
null を返す。

extern BDD operator|(const BDD& f, const BDD& g)
f と g の論理和を表す BDD オブジェクトを生成し、それを返す。記憶あふれの
場合は、null を表すオブジェクトを返す。引数に null を与えた場合には
null を返す。

extern BDD operator^(const BDD&, const BDD&)
f と g の排他的論理和を表す BDD オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた場合
には null を返す。

extern int operator==(const BDD& f, const BDD& g)
f と g が同じ論理関数かどうかの真偽(1/0)を返す。

extern int operator!=(const BDD& f, const BDD& g)
f と g が異なる論理関数かどうかの真偽(1/0)を返す。

extern int BDD_Imply(BDD f, BDD g)
f と g の包含性判定を行う。すなわち、(~f | g) が恒真かどうかを調べる。
(~f | g) の BDD オブジェクトを生成せずに判定だけを行うので、(~f | g)の
演算を実行するよりも高速である。引数に null を与えた場合には 0 を返す。
Extern BDD BDD_Import(FILE *strm = stdin)
strm で指定するファイルから BDD の構造を読み込み、BDD オブジェクトを生成して、それ
を返す。ただし、ファイルに書かれているデータが多出力であった場合は、最初の出力の
論理関数のみ読み込む。ファイルに文法誤りが合った場合等、異常終了時は null を返す。

extern BDD BDD_Random(int dim, int density = 50)
入力数 dim（変数の level が 1 から dim まで）の乱数論理関数（真理値表が
乱数表である論理関数）を表す BDD オブジェクトを生成し、それを返す。
引数 density によって、真理値表濃度（１の出現確率%）を指定することが
できる。記憶あふれの場合は、null を表すオブジェクトを返す。

extern void BDDerr(char *msg)
extern void BDDerr(char *msg, bddword key)
extern void BDDerr(char *msg, char *name)
引数として与えた文字列や数値をエラー出力に表示し、異常終了する。
通常、内部で回復不可能なエラーが起きたときに自動的に呼び出されるが、
何らかの理由で処理を途中終了したいときに陽に用いても良い。

----------公開クラスメソッド----------
BDD::BDD(void)
基本 constructer。初期値として恒偽関数を表す BDD オブジェクトを生成する。

BDD::BDD(int val)
定数論理関数のオブジェクトを作り出す constructer。val == 0 ならば恒偽関数、
val > 0 ならば恒真関数、val < 0 ならば null を表す BDD オブジェクトを生成する。

BDD::BDD(const BDD& f)
引数 f を複製する constructer。

BDD::~BDD(void)
destructer。内部の BDD 節点の記憶管理は自動化されており、使用済みの節点は適当なタ
イミングで回収され、再利用される。

BDD& BDD::operator=(const BDD& f)
自分自身に f を代入し、f を関数値として返す。

BDD BDD::operator&=(const BDD& f)
自分自身と f との論理積を求め、自分自身に代入する。演算結果を関数値として返す。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは何もし
ない。f が null のときは、null を代入する。

BDD BDD::operator|=(const BDD& f)
自分自身と f との論理和を求め、自分自身に代入する。演算結果を関数値として返す。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは何もし
ない。f が null のときは、null を代入する。

BDD BDD::operator^=(const BDD& f)
自分自身と f との排他的論理和を求め、自分自身に代入する。演算結果を関数値
として返す。記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が
null のときは何もしない。f が null のときは、null を代入する。

BDD BDD::operator<<=(int s)
自分自身のグラフに対して、関係する全ての入力変数を展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて BDD を複製した
論理関数を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を指定する
ことはできない。

BDD BDD::operator>>=(int s)
自分自身のグラフに対して、関係する全ての入力変数を展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて BDD を複製した
論理関数を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
したがって、f に関係しない入力変数が下位レベルにあらかじめ用意されていなければ
ならない。記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null の
ときは何もしない。s に負の値を指定することはできない。

BDD BDD::operator~(void)
自分自身の否定の論理関数を表す BDD オブジェクトを生成し、それを返す。
自分自身が null のときは、null を返す。

BDD BDD::operator<<(int s)
自分自身のグラフに対して、関係する全ての入力変数を展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて BDD を複製した
オブジェクトを生成し、それを返す。実行結果において未定義の入力変数が必要に
なるような s を与えてはならない。必要な入力変数はあらかじめ宣言しておくこと。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは
何もしない。s に負の値を指定することはできない。

BDD BDD::operator>>(int)
自分自身のグラフに対して、関係する全ての入力変数を展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて BDD を複製した
オブジェクトを生成し、それを返す。実行結果において未定義の入力変数が必要
になるような s を与えてはならない。したがって、f に関係しない入力変数が
下位レベルにあらかじめ用意されていなければならない。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

BDD BDD::At0(int var)
自分自身のグラフに対して、番号 var の入力変数を 0 に固定したときの
論理関数（射影）を表す BDD オブジェクトを生成し、それを返す。記憶あふれ
の場合は、null を表すオブジェクトを返す。自分自身が null のときは、
null を返す。

BDD BDD::At1(int var)
自分自身のグラフに対して、番号 var の入力変数を 1 に固定したときの
論理関数（射影）を表す BDD オブジェクトを生成し、それを返す。記憶あふれ
の場合は、null を表すオブジェクトを返す。自分自身が null のときは、
null を返す。

BDD BDD::Cofact(BDD f)
自分自身のグラフに対して、f = 0 の部分を don't care とみなして簡単化を
行った論理関数を表す BDD オブジェクトを生成し、それを返す。記憶あふれの場合
は、null を表すオブジェクトを返す。自分自身が null のとき、および f が
null のときは、null を返す。

BDD BDD::Univ(BDD f)
全称作用演算(universal quantification)。f で指定した入力変数の部分集合
に 0,1 の定数を代入したときに、どのような 0,1 の組合せを代入しても常に自分
自身が 1 となる場合には 1 を返し、それ以外は 0 を返すような論理関数を表すオ
ブジェクトを生成し、それを返す。入力変数の部分集合の指定は、それらの変
数すべての論理和を表す論理関数を作って与える。記憶あふれの場合は、null
を表すオブジェクトを返す。自分自身が null のとき、および f が null のと
きは、null を返す。

BDD BDD::Exist(BDD f)
存在作用演算(universal quantification)。f で指定した入力変数の部分集合
に 0,1 の定数を代入したときに、どのような 0,1 の組合せを代入しても常に自分
自身が 0 となる場合には 0 を返し、それ以外は 1 を返すような論理関数を表すオ
ブジェクトを生成し、それを返す。入力変数の部分集合の指定は、それらの変
数すべての論理和を表す論理関数を作って与える。記憶あふれの場合は、null
を表すオブジェクトを返す。自分自身が null のとき、および f が null のと
きは、null を返す。

BDD BDD::Support(void)
自分自身の論理関数の値に影響を与える入力変数の集合を抽出し、それらの変
数すべての論理和を表すオブジェクトを作りそれを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null のとき、および f が null
のときは、null を返す。

int BDD::Top(void)
自分自身のグラフに対して、最上位の入力変数の番号を返す。定数関数または
null のときは、0 を返す。

bddword BDD::Size(void)
自分自身のグラフの節点数を返す。null に対しては 0 を返す。

void BDD::Export(FILE *strm = stdout)
BDD の内部データ構造を、strm で指定するファイルに出力する。

void BDD::XPrint0(void)
自分自身のグラフを、X-Window に描画する。(否定エッジなし)

void BDD::XPrint(void)
自分自身のグラフを、X-Window に描画する。(否定エッジなし)

bddword BDD::GetID(void)
論理関数を一意に表現する 1-word の識別番号（内部インデックス値）を返す。

void BDD::Print(void)
BDD の内部インデックス値、最上位の変数番号、節点数の情報を標準出力に出力する。

BDD BDD::Swap(int var1, int var2)
自分自身のグラフに対して、変数番号 var1 と var2 の入力変数を
入れ換えたときの論理関数を表すオブジェクトを生成し、それを返す。
引数は level ではなく、変数番号で与えることに注意。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null のときは、null を返す。

BDD BDD::Smooth(int var)
自分自身のグラフに対して、指定した変数番号 var の順位よりも低い順位を持つ
全ての入力変数に、あらゆる 0,1 の組合せを代入したときに、関数の値が真になる
組合せが 1 つでもあるか否かを表す BDD を生成し、それを返す。計算結果の BDD
には var およびそれ以下の順位を持つ変数の節点は含まれない。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは、null を返す。

****
クラス名: BDDV --- BDD の配列（論理関数の配列）を表すクラス
****
ヘッダーファイル名: "BDD.h"
ソースファイル名: BDD.cc
内部から呼び出しているクラス: BDD

BDD の配列を表すクラスである。配列長は可変である。要素の番号は 0 か
ら始まる整数である。内部構造は、「出力選択変数」を導入して、
１個の BDD に束ねて配列を表現しているので、もし配列長が大きくても、
各要素が同じ関数であれば、メモリ使用量は１個の BDD と変わらない。
また、配列の複製が配列長に関わらず定数時間で行える。
BDDV_Init()を実行すると、入力変数番号 1 から BDDV_SysVarTop(通常 20) までの
変数が出力選択変数としてシステムに確保され、ユーザが論理演算に用いる
入力変数はその次の番号(通常 21)から始まる。出力選択変数はユーザ変数よりも
必ず上位になるように自動的に配置される。

----------関連する定数値----------
extern const int BDDV_SysVarTop
出力選択変数の個数。通常 20 である。ユーザの使用する論理変数の
番号は(BDDV_SysVarTop + 1)番から始まる。

extern const int BDDV_MaxLen
配列の最大長。BDDV_SysVarTop のべき乗である。

extern const bddword BDD_MaxNode
extern const int BDD_MaxVar
----------関連する外部関数----------
extern void   BDDV_Init(bddword init, bddword limit)
BDD_Init()と同様に、処理系を初期化しメモリの確保を行う。BDD_Init()との
相違点は、出力選択変数の確保を行う点である。入力変数番号 1 から
BDDV_SysVarTop(通常 20) までの変数が、出力選択変数としてシステムに
確保され、ユーザが論理演算に用いる入力変数はその次の番号(通常 21)から
始まる。出力選択変数はユーザ変数よりも必ず上位になるように自動的に
配置される。BDDV を用いる場合は、必ず最初に BDDV_Init()を実行しなければ
ならない。

extern int BDDV_UserTopLev(void)
（このメソッドは旧版で用いていた。なくても困らないはずである）
現在までにユーザが宣言した論理変数の個数。出力選択変数は含まれない。
この数値は、現在までにユーザが宣言した論理変数の順位(level)の最上位の
順位番号と等しい。出力選択変数の順位番号(level)は、そのすぐ上の範囲に
あり、(BDDV_UserTopLev + 1)から(BDDV_UserTopLev + BDDV_SysVarTop)までの
間である。

extern int BDDV_NewVar(void)
（このメソッドは旧版で用いていた。なくても困らないはずである）
BDD_NewVar()と同様に、新しい入力変数を１つ生成し、その変数番号(通称 VarID)を
返す。BDD_NewVar()との違いは、VarID が 1 から始まるのではなく、
(BDDV_SysVarTop + 1)から始まる点である。ただし変数の順位(通称 level)は 1 から
スタートする。出力選択変数の level は 1 ずつ上位にシフトしていく。

extern int BDDV_NewVarOfLev(int lev)
（このメソッドは旧版で用いていた。なくても困らないはずである）
BDD_NewVarOfLev()と同様に、順位（通称 level）を指定して新しい入力変数を
１つ生成し、その変数番号(通称 VarID)を返す。BDD_NewVar()との違いは、
VarID が 1 から始まるのではなく、(BDDV_SysVarTop + 1)から始まる点である。
ただし指定できる変数の順位(通称 level)は、1 以上かつ「これまでユーザが宣言
した変数の個数 +1」までである。出力選択変数の level は 1 ずつ上位にシフトしていく。

extern BDDV operator&(const BDDV& fv, const BDDV& gv)
fv と gv の各要素同士の論理積を表す BDDV オブジェクトを生成し、それを返す。
配列長が一致していなければエラー（異常終了）。記憶あふれの場合は 長さ 1 の null を
返す。引数に null が含まれていた場合には、長さ 1 の null を返す。

extern BDDV operator|(const BDDV& fv, const BDDV& gv)

fv と gv の各要素同士の論理和を表す BDDV オブジェクトを生成し、それを返す。
配列長が一致していなければエラー（異常終了）。記憶あふれの場合は 長さ１の null を
返す。引数に null が含まれていた場合には、長さ１の null を返す。

extern BDDV operator^(const BDDV& fv, const BDDV& gv)
fv と gv の各要素同士の排他的論理和を表す BDDV オブジェクトを生成し、それを返す。
配列長が一致していなければエラー（異常終了）。記憶あふれの場合は 長さ１の null を
返す。引数に null が含まれていた場合には、長さ１の null を返す。

extern int operator==(const BDDV& fv, const BDDV& gv)
fv と gv の対応する要素が全て同じ論理関数かどうかの真偽(1/0)を返す。
配列長が一致していなければエラー（異常終了）。

extern int operator!=(const BDDV& fv, const BDDV& gv)
fv と gv の対応する要素の少なくとも１組が異なる論理関数かどうかの
真偽(1/0)を返す。配列長が一致していなければエラー（異常終了）。

extern int BDDV_Imply(BDDV fv, BDDV gv)
fv と gv の包含性判定を行う。すなわち、(~fv | gv) が全ての要素について
恒真かどうかを調べる。(~fv | gv) の BDDV オブジェクトを生成せずに
判定だけを行うので、(~fv | gv)の演算を実行するよりも高速である。
引数に null を与えた場合には 0 を返す。

Extern BDDV BDDV_Import(FILE *strm = stdin)
strm で指定するファイルから BDDV の構造を読み込み、BDDV オブジェクトを生成して、そ
れを返す。ファイルに文法誤りが合った場合等、異常終了時は null を返す。

extern BDDV operator||(const BDDV& fv, const BDDV& gv)
fv の末尾に gv を連結した BDDV オブジェクトを生成し、それを返す。
fv, gv は変化しない。fv の配列長が２のべき乗数のとき、処理効率がよい。
記憶あふれの場合は 長さ１の null を返す。引数に null が含まれていた場合には、
長さ１の null を返す。

extern BDDV BDDV_Mask1(int ix, int len)
ix 番目の要素だけが恒真関数で、他は恒偽関数となっているような、長さ
len の多出力定数論理関数を表す BDDV オブジェクトを生成し、それを返す。

記憶あふれの場合は 長さ１の null を返す。引数に null が含まれていた場合には、
長さ１の null を返す。不当な引数を与えた場合はエラー（異常終了）。

extern BDDV BDDV_Mask2(int ix, int len)
0 番目～(ix-1) 番目の要素が恒偽関数で、ix 番目以降は恒真関数となっているような、
長さ len の多出力定数論理関数を表す BDDV オブジェクトを生成し、それを返す。
記憶あふれの場合は 長さ１の null を返す。引数に null が含まれていた場合には、
長さ１の null を返す。不当な引数を与えた場合はエラー（異常終了）。

(再掲)
extern int BDD_NewVar(void)
extern int BDD_NewVarOfLev(int lev)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)

--------------------公開クラスメソッド--------------------
BDDV::BDDV(void)
基本 constructer。配列長 0 の BDDV オブジェクトを生成する。

BDDV::BDDV(const BDDV& fv)
引数 fv を複製する constructer。

BDDV::BDDV(const BDD&, int len = 1)
引数 f で指定した BDD が len 個続けて並んでいる BDDV オブジェクトを
生成する constructer。f に null を与えた場合は、len 指定に関わらず
長さ１となる。

BDDV::~BDDV(void)
destructer。

BDDV& BDDV::operator=(const BDDV& fv)
自分自身の元のデータを消去し、fv を代入する。関数の値として fv を返す。

BDDV BDDV::operator&=(const BDDV& fv)
自分自身と fv の各要素同士の論理積を求め、自分自身に代入する。配列長は一致
していなければならない。記憶あふれの場合は 長さ１の null を返す。自分自身
または引数に null が含まれていた場合には、長さ１の null を返す。

BDDV BDDV::operator|=(const BDDV& fv)
自分自身と fv の各要素同士の論理和を求め、自分自身に代入する。配列長は一致
していなければならない。記憶あふれの場合は 長さ１の null を返す。自分自身
または引数に null が含まれていた場合には、長さ１の null を返す。

BDDV BDDV::operator^=(const BDDV& fv)
自分自身と fv の各要素同士の排他的論理和を求め、自分自身に代入する。配列長は
一致していなければならない。記憶あふれの場合は 長さ１の null を返す。自分自身
または引数に null が含まれていた場合には、長さ１の null を返す。

BDDV BDDV::operator<<=(int s)
自分自身の各要素に対して、関係する全ての入力変数を展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
BDDV を、自分自身に代入する。また演算結果を関数値として返す。出力選択変数には
影響はない。実行結果において未定義の入力変数が必要になるような s を与えて
はならない。必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null のときは何もしない。s に負の値を
指定することはできない。

BDDV BDDV::operator>>=(int s)
自分自身の各要素に対して、関係する全ての入力変数を展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
BDDV を、自分自身に代入する。また演算結果を関数値として返す。出力選択変数には
影響はない。実行結果において未定義の入力変数が必要になるような s を与えて
はならない。必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null のときは何もしない。s に負の値を
指定することはできない。

BDDV BDDV::operator~(void)
自分自身の各要素の否定関数を表す BDDV オブジェクトを生成し、それを返す。

自分自身に null が含まれていた場合には、長さ１の null を返す。

BDDV BDDV::operator<<(int s)
自分自身の各要素に対して、関係する全ての入力変数を展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
BDDV を生成し、それを返す。出力選択変数には影響はない。実行結果において
未定義の入力変数が必要になるような s を与えてはならない。必要な入力変数は
あらかじめ宣言しておくこと。記憶あふれの場合は、null を表すオブジェクトを返す。
自分自身が null のときは何もしない。s に負の値を指定することはできない。

BDDV BDDV::operator>>(int s)
自分自身の各要素に対して、関係する全ての入力変数を展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
BDDV を生成し、それを返す。出力選択変数には影響はない。実行結果において
未定義の入力変数が必要になるような s を与えてはならない。必要な入力変数は
あらかじめ宣言しておくこと。記憶あふれの場合は、null を表すオブジェクトを返す。
自分自身が null のときは何もしない。s に負の値を指定することはできない。

BDDV BDDV::At0(int var)
自分自身の各要素に対して、変数番号 var の入力変数を 0 に固定したときの
論理関数（射影）を表すオブジェクトを生成し、それを返す。記憶あふれの場
合は 長さ１の null を返す。自分自身に null が含まれていた場合には、長さ１の
null を返す。不当な引数を与えた場合はエラー（異常終了）となる。

BDDV BDDV::At1(int var)
自分自身の各要素に対して、変数番号 var の入力変数を 1 に固定したときの
論理関数（射影）を表すオブジェクトを生成し、それを返す。記憶あふれの場
合は 長さ１の null を返す。自分自身に null が含まれていた場合には、長さ１の
null を返す。不当な引数を与えた場合はエラー（異常終了）となる。

BDDV BDDV::Cofact(BDDV fv)
自分自身の各要素に対して、fv = 0 の部分を don't care とみなして簡単化を
行った論理関数を表す BDDV オブジェクトを生成し、それを返す。記憶あふれの場合は
長さ１の null を返す。自分自身または引数に null が含まれていた場合には、
長さ１の null を返す。

BDDV BDDV::Swap(int var1, int var2)
自分自身の各要素に対して、番号 var1 と var2 の入力変数を入れ換えた
ときの論理関数を表す BDDV オブジェクトを生成し、それを返す。level ではなく
VarID を指定することに注意。記憶あふれの場合は 長さ１の null を返す。自分
自身に null が含まれていた場合には、長さ１の null を返す。不当な引数を与えた
場合はエラー（異常終了）となる。

int BDDV::Top(void)
自分自身の各要素に関して、最上位の入力変数の番号（各要素の中の最大値）
を返す。全要素が定数関数のときは０を返す。自分自身に null が含まれてい
た場合には、０を返す。

bddword BDDV::Size(void)
自分自身の BDDV の節点数を返す。各要素の BDD は互いにサブグラフを共有する
ので、各要素のサイズの総和より小さいことがある。自分自身に null が含まれて
いた場合には、０を返す。出力選択変数に関する節点は含まない。

void BDDV::Export(FILE *strm = stdout)
BDDV の内部データ構造を、strm で指定するファイルに出力する。

void BDDV::XPrint0(void)
自分自身のグラフを、X-Window に描画する。(否定エッジなし)

void BDDV::XPrint(void)
自分自身のグラフを、X-Window に描画する。(否定エッジあり)

BDDV BDDV::Former(void)
自分自身の前半部分列を表す BDDV オブジェクトを生成し、それを返す。
前半部分列の長さは、元の配列長より小さな、最大の２のべき乗数である。
ただし、元の長さが１以下のときは、前半部分列の長さは０とする。
自分自身に null が含まれていた場合には、長さ１の null を返す。
不当な引数を与えた場合はエラー（異常終了）となる。

BDDV BDDV::Latter(void)
自分自身の後半部分列を表す BDDV オブジェクトを生成し、それを返す。
後半部分列は、前半部分列に含まれない部分のことを言う。記憶あふれの
場合は 長さ１の null を返す。自分自身に null が含まれていた場合には、
長さ１の null を返す。不当な引数を与えた場合はエラー（異常終了）となる。

BDDV BDDV::Part(int start, int len)
自分自身の任意の部分列を表すオブジェクトを生成し、それを返す。start 番
目の要素から始まる len 個の要素を取り出す。記憶あふれの場合は 長さ１の null を
返す。自分自身に null が含まれていた場合には、長さ１の null を返す。不当な引数を
与えた場合はエラー（異常終了）となる。

BDD BDDV::GetBDD(int ix)
第 ix 番目の要素の BDD を表すオブジェクトを生成し、それを返す。
不当な ix を与えた場合はエラー（異常終了）となる。

BDD BDDV::GetMetaBDD(void)
内部構造の BDD そのもの（出力選択変数も含む）を表すオブジェクトを
生成し、それを返す。

int BDDV::Uniform(void)
全要素が同じ要素であるかどうかの真偽（同じならば１）を返す。

int BDDV::Len(void)
配列長を返す。

void BDDV::Print(void)
内部情報を標準出力に出力する。

****
クラス名: Btol --- ２値入力整数値出力の論理関数を表すクラス
****
ヘッダーファイル名: "Btol.h"
ソースファイル名: Btol.cc

内部から呼び出しているクラス: BDD, BDDV

整数値論理関数データを表すクラス。整数値論理関数は２値の論理ベクトルから
整数への関数である。内部データは整数値を２進数で符号化することにより、
２値の論理関数のベクトルとして、BDDV を用いて表現されている。負数は、
２の補数表現を採用している。２進数の桁数は、関数が取りうる最大値に
合わせて、常に自動的に調整される。関数が０から-1 の範囲にあれば桁数は
１となり、1 から-2 の範囲にあれば２ビット、3 から-4 の範囲なら３ビット、
127 から-128 の範囲では８ビットとなる。桁数の上限は約 100 万（ただし約 100
を超えると多倍長計算を行うため処理速度が低下する)。最上位ビットは正負の条件を
表す(負数のときに１)。記憶あふれの場合は、Btol(BDD(-1))を返す(以下、null と呼ぶ)。

----------関連する定数値----------
extern const int BDDV_SysVarTop
extern const int BDDV_MaxLen
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar

----------関連する外部関数----------
extern int operator==(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv が同じ整数値論理関数を表すならば 1、
そうでなければ 0 を返す。Btol_EQ() とは意味が違うので注意。

extern int operator!=(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv が同じ整数値論理関数を表すならば 0、
そうでなければ 1 を返す。Btol_NE() とは意味が違うので注意。

extern Btol operator+(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv の算術和を表す Btol オブジェクトを新しく生成
し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。

extern Btol operator-(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv の算術差を表す Btol オブジェクトを新しく生成
し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。

extern Btol operator*(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv の算術積を表す Btol オブジェクトを新しく生成
し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。

extern Btol operator/(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv の整数除算の商を表す Btol オブジェクトを新し
く生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null
が含まれる場合は null を返す。結果が負で割り切れないときは、絶対値の小さ
い方に切り捨てる。(例) 10/(-3)= -3

extern Btol operator%(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv の算術和を表す Btol オブジェクトを新しく生成
し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。商が負になる場合でも、fv = gv * (fv / gv) + (fv % gv)
を満たすように計算する。(例) -10 % 3 = -1

extern Btol operator&(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv のビット論理積を表す Btol オブジェクトを新し
く生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null
が含まれる場合は null を返す。

extern Btol operator|(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv のビット論理和を表す Btol オブジェクトを新し
く生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null
が含まれる場合は null を返す。

extern Btol operator^(const Btol& fv, const Btol& gv)
２つの Btol オブジェクト fv, gv のビット排他論理和を表す Btol オブジェクトを
新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に
null が含まれる場合は null を返す。

extern Btol Btol_ITE(BDD f, Btol gv, Btol hv)
f が１のときは gv を、f が０のときは hv を返すような整数値論理関数を表す
Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は

null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_ITE(Btol fv, Btol gv, Btol hv)
fv が 0 以外のときは gv を、fv が 0 のときは hv を返すような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_EQ(Btol fv, Btol gv)
fv と gv が同じ出力値となるような入力組合せに対して 1、そうでないとき 0 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。operator == とは意味が違うので注意。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_NE(Btol fv, Btol gv)
fv と gv が同じ出力値となるような入力組合せに対して 0、そうでないとき 1 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。operator != とは意味が違うので注意。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_GT(Btol fv, Btol gv)
fv が gv より大きい値となるような入力組合せに対して 1、そうでないとき 0 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_LT(Btol fv, Btol gv)
fv が gv より小さい値となるような入力組合せに対して 1、そうでないとき 0 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_GE(Btol fv, Btol gv)
fv が gv より大きいか等しい値となるような入力組合せに対して 1、そうでないとき 0 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_LE(Btol fv, Btol gv)
fv が gv より小さいか等しい値となるような入力組合せに対して 1、そうでないとき 0 となるような整数値論理関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Btol Btol_atoi(char* s)
s の指す数字文字列を数値に変換し、定数関数の Btol オブジェクトを新しく生成し、それを値として返す。通常は 10 進数として変換するが、文字列の先頭が ``0X'' または ``0x'' で始まる場合には 16 進数、 ``0B'' または ``0b'' で始まる場合は 2 進数で変換する。それ以外の不適当な文字を含む文字列に対する動作は保証しない。

（再掲）
extern void BDDV_Init(bddword init, bddword limit)
extern int BDD_NewVar(void)
extern int BDD_NewVarOfLev(int lev)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)

----------公開クラスメソッド----------
Btol::Btol(void)
基本 constructer。初期値として恒偽関数（常に 0 を返す）を表すオブジェクトを生成する。

Btol::Btol(const Btol& fv)
Btol オブジェクトのコピーを生成する constructer。

Btol::Btol(const BDD& f)
論理関数 f の真偽にしたがって、整数値 1/0 を返す整数値論理関数を表す Btol オブジェクトを生成する constructer。f が null の場合は 1 ビットの null からなるオブジェクト Btol(BDD(-1)) を生成。

Btol::Btol(int n)
定数関数（常に整数値 n を返す）を表す Btol オブジェクトを生成する constructer。

Btol::Btol(const BDDV& fv)
自分自身の内部データの BDDV オブジェクト fv を与える constructor。

Btol::~Btol(void)
destructer。

Btol& Btol::operator=(const Btol& fv)
fv を自分自身に代入し、それを値として返す。

Btol Btol::operator+=(const Btol& fv)
自分自身と fv の算術和を自分自身に代入し、それを値として返す。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator-=(const Btol& fv)
自分自身と fv の算術差を自分自身に代入し、それを値として返す。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator*=(const Btol& fv)
自分自身と fv の算術積を自分自身に代入し、それを値として返す。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator/=(const Btol& fv)
自分自身と fv の整数除算の商を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。結果が負で割り切れないときは、絶対値の小さい方に切り捨てる。(例)
10/(-3)= -3

Btol Btol::operator%=(const Btol& fv)
自分自身とvの整数除算の剰余を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。商が負になる場合でも商の絶対値が小さくなるように剰余を計算する。

（例）-10 % 3 = -1

Btol Btol::operator&=(const Btol& fv)
自分自身と fv のビット論理積を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator|=(const Btol& fv)
自分自身と fv のビット論理和を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator^=(const Btol& fv)
自分自身と fv のビット排他論理和を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator<<=(const Btol& fv)
自分自身を、fv が表すビット数だけ算術的に左シフトした結果を自分自身に代入し、それを値として返す。fv が負のときは右にシフトする。0 のときはシフトしない。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator>>=(const Btol& fv)
自分自身を、fv が表すビット数だけ算術的に右シフトした結果を自分自身に代入し、それを値として返す。vが負のときは左にシフトする。0 のときはシフトしない。自分自身が負数のときは、左端のビットには 1 が入る。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator-(void)
自分自身の補数を表す Btol オブジェクトを新しく生成し、それを値として返す。
記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::operator~(void)
自分自身のビット反転（1 の補数）を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::operator!(void)

自分自身の論理否定を表す Btol オブジェクトを新しく生成し、それを値として返す。論理否定は 0 に対して 1 を返し、0 以外の値に対しては 0 を返す演算である。Btol_EQ(*this, 0)と等価。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::operator<<(const Btol& fv)
自分自身を、fv が表すビット数だけ算術的に左シフトした結果を表す Btol オブジェクトを新しく生成し、それを値として返す。vが負のときは右にシフトする。0 のときはシフトしない。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::operator>>(const Btol& fv)
自分自身を、fv が表すビット数だけ算術的に右シフトした結果を表す Btol オブジェクトを新しく生成し、それを値として返す。vが負のときは左にシフトする。0 のときはシフトしない。自分自身が負数のときは、左端のビットには 1 が入る。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

Btol Btol::UpperBound(void)
考えられるすべての入力組合せについて、自分自身が取り得る最大値を表す定数関数の Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::UpperBound(BDD f)
f で指定した入力変数の部分集合に対するすべての入力組合せについて、自分自身が取り得る最大値を表す Btol オブジェクトを新しく生成し、それを値として返す。入力変数の部分集合 f は、入力変数の論理和の形式で与える。
記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::LowerBound(void)
考えられるすべての入力組合せについて、自分自身が取り得る最小値を表す定数関数の Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::LowerBound(BDD)
f で指定した入力変数の部分集合に対するすべての入力組合せについて、自分自身が取り得る最小値を表す Btol オブジェクトを新しく生成し、それを値として返す。入力変数の部分集合 f は、入力変数の論理和の形式で与える。
記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::At0(int var)
自分自身の表す関数において、変数番号 var で指定した入力変数に 0 を代入した関数を表す Btol オブジェクトを新しく生成し、それを値として返す。不当な var を与えた場合はエラー（異常終了）。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::At1(int)
自分自身の表す関数において、変数番号 var で指定した入力変数に 1 を代入した関数を表す Btol オブジェクトを新しく生成し、それを値として返す。不当な var を与えた場合はエラー（異常終了）。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Btol Btol::Cofact(Btol fv)
自分自身の表す関数において、fv = 0 のときを don't care とみなして簡単化した関数を表す Btol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返す。

int Btol::Top(void)
自分自身の表す関数に関係する入力変数の中で、最高の順位(level)を持つ変数の番号(VarID)を返す。level の値そのものを返すわけではないので注意。
定数関数のときは 0 を返す。null のときは 0 を返す。

BDD Btol::GetSignBDD(void)
自分自身の表す関数の符号ビット（負になるための条件）を表す BDD オブジェクトを新しく生成し、それを値として返す。null の場合は null の BDD を返す。

BDD Btol::GetBDD(int ix)
自分自身の表す関数の第 ix ビットを表す BDD オブジェクトを新しく生成し、
それを値として返す。ix は最下位ビットから 0, 1, 2, ... となっている。
負の ix を与えた場合はエラー。桁数を超える ix に対しては、最上位（符号）ビットの BDD を返す。自分自身が null の場合は null の BDD を返す。

BDDV Btol::GetMetaBDDV(void)
自分自身の内部データの BDDV オブジェクトをコピーして返す。

int Btol::Len(void)
自分自身の桁数を返す。

int Btol::GetInt(void)
自分自身の表す関数が定数関数の場合、その整数値を返す。定数関数でない場合は、すべての入力変数に 0 を代入したときの値を出力する。整数値が 32 ビットに収まらない場合は、下位 32 ビットのみを有効とする。null の場合は 0 を返す。

int Btol::StrNum10(char* s)
自分自身が表す関数が定数関数の場合、その 10 進数文字列を生成し、s から始まる領域に書き込む。s には事前に十分な領域（最低 10 バイト）を確保しておかなければならない。定数関数でない場合は、すべての入力変数に 0 を代入したときの値を出力する。関数の値は通常 0 を返すが、記憶あふれの場合や、null のときは 1 を返す。このとき文字列には"0"が入る。

int Btol::StrNum16(char* s)
自分自身が表す関数が定数関数の場合、その 16 進数文字列を生成し、s から始まる領域に書き込む。s には事前に十分な領域（最低 8 バイト）を確保しておかなければならない。定数関数でない場合は、すべての入力変数に 0 を代入したときの値を出力する。関数の値は通常 0 を返すが、記憶あふれの場合や、null のときは 1 を返す。このとき文字列には"0"が入る。

bddword Btol::Size()
自分自身のグラフの節点数を返す。null の場合は 0 を返す。

void Btol::Print(void)
自分自身の内部データ情報を標準出力に出力する。

****
クラス名: ZBDD --- ゼロサプレス型 BDD で表現された組合せ集合を指すクラス
****
ヘッダーファイル名: "ZBDD.h"
ソースファイル名: ZBDD.cc
内部から呼び出しているクラス: BDD

ゼロサプレス型 BDD 表現を用いて、組合せ集合を抽象化したクラスである。
集合の各要素は、n 個のアイテムの中から k 個を選ぶ組合せを表す。アイテムは 1 から始まる整数で識別する。0 個の要素からなる集合を空集合と呼ぶ。また n 個のリテラルから 0 個を選ぶ組合せを表す要素を単位元要素と呼び、単位元要素 1 個だけからなる集合を単位元集合と呼ぶ。記憶あふれの場合は処理を中断し、null (-1) を返す。
（使用例）
```
ZBDD x = ZBDD(1).Change(1);
ZBDD y = ZBDD(1).Change(2);
ZBDD f = x + y;
ZBDD g = f.Change(3) + f.Change(4);
f.Print();
g.Print();
```

----------関連する定数値----------
```
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar
```

----------関連する外部関数----------
```
extern ZBDD operator&(const ZBDD& f, const ZBDD& g)
```
f と g の交わり(intersection)を表す ZBDD オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた場合には null を返す。

```
extern ZBDD operator+(const ZBDD& f, const ZBDD& g)
```
f と g の結び(union)を表す ZBDD オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた

場合には null を返す。

extern ZBDD operator-(const ZBDD& f, const ZBDD& g)
f から g を引いた差分集合を表す ZBDD オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた
場合には null を返す。

extern ZBDD operator*(const ZBDD& f, const ZBDD& g)
f と g の直積集合を表す ZBDD オブジェクトを生成し、それを返す。記憶
あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた
場合には null を返す。

extern ZBDD operator/(const ZBDD& f, const ZBDD& g)
f を g で割った集合(Weak-division)を表す ZBDD オブジェクトを生成し、
それを返す。記憶あふれの場合は、null を表すオブジェクトを返す。
引数に null を与えた場合には null を返す。

extern ZBDD operator%(const ZBDD& f, const ZBDD& g);
f を g で割った剰余の集合を表す ZBDD オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた
場合には null を返す。

extern int operator==(const ZBDD& f, const ZBDD& g)
f と g が同じ集合かどうかの真偽(1/0)を返す。

extern int operator!=(const ZBDD& f, const ZBDD& g)
f と g が異なる集合かどうかの真偽(1/0)を返す。

extern ZBDD BDD_CacheZBDD(char op, bddword f, bddword g);
f と g の演算結果が ZBDD 型のとき、演算結果をキャッシュから参照する。op
は演算の種類を表す番号で、20 以上の値を入れる。演算結果が登録されて
いる場合はその ZBDD を返し、見つからなかった場合は、null を表すオブジェ
クトを返す。f, g が ZBDD 型の演算の場合は、GetID()で bddword 型に変換して
与える。

Extern ZBDD ZBDD_Import(FILE *strm = stdin)
strm で指定するファイルから ZBDD の構造を読み込み、ZBDD オブジェクトを生成して、そ
れを返す。ただし、ファイルに書かれているデータが多出力であった場合は、最初の出力
の構造のみ読み込む。ファイルに文法誤りが合った場合等、異常終了時は null を返す。

extern ZBDD ZBDD_Random(int dim, int density = 50)
次数 dim の乱数集合を表す ZBDD オブジェクトを生成し、それを返す。
変数順位(level)が 1 から dim までの値を持つアイテム変数を使用する。
アイテム変数はあらかじめ宣言されていなければならない。density によって、
濃度（要素数／全体集合%）を指定することができる。記憶あふれの場合は
null を返す。

extern ZBDD_Meet(const ZBDD& f, const ZBDD& g)
f と g の Meet 演算（Knuth 本 4 巻 1 分冊 141 頁: 演習問題 203 参照）により得られる
集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合は、null を
表すオブジェクトを返す。引数に null を与えた場合には null を返す。

（再掲）
void BDD_Init(bddword init, bddword limit)
extern int BDD_NewVar(void)
extern int BDD_NewVarOfLev(int lev)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)

----------公開クラスメソッド----------
ZBDD::ZBDD(void)
constructer。初期値として空集合を表すオブジェクトを生成する。

ZBDD::ZBDD(int val)
定数を表す ZBDD オブジェクトを作り出す constructer。v == 0 ならば
空集合、v > 0 ならば単位元集合、v < 0 ならば null を表すオブジェクトを
生成する。

ZBDD::ZBDD(const ZBDD& f)
引数 f を複製する constructer。

ZBDD::~ZBDD(void)
destructer。

ZBDD& ZBDD::operator=(const ZBDD& f)
自分自身に f を代入し、そのコピーを返す。

ZBDD ZBDD::operator&=(const ZBDD& f)
自分自身と f との交わり(intersection)を求め、自分自身に代入し、
そのコピーを返す。記憶あふれの場合は、null を表すオブジェクトを
代入する。自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator+=(const ZBDD& f)
自分自身と f との結び(union)を求め、自分自身に代入し、そのコピーを
返す。記憶あふれの場合は、null を表すオブジェクトを代入する。
自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator-=(const ZBDD& f)
自分自身から f を引いた差分集合を求め、自分自身に代入し、そのコピーを
返す。記憶あふれの場合は、null を表すオブジェクトを代入する。
自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator*=(const ZBDD& f)
自分自身と f との直積集合を求め、自分自身に代入し、そのコピーを返す。
記憶あふれの場合は、null を表すオブジェクトを代入する。
自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator/=(const ZBDD& f)
自分自身を f で割った集合(Weak division)を求め、自分自身に代入し、
そのコピーを返す。記憶あふれの場合は、null を表すオブジェクトを代入する。
自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator%=(const ZBDD& f)
自分自身を f で割った余りの集合を求め、自分自身に代入し、そのコピーを返す。
記憶あふれの場合は、null を表すオブジェクトを代入する。
自分自身または引数が null の場合も null となる。

ZBDD ZBDD::operator<<=(int s)
自分自身のグラフに対して、関係する全てのアイテム変数を、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて ZBDD を複製した
組合せ集合を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を指定する
ことはできない。

ZBDD ZBDD::operator>>=(int)
自分自身のグラフに対して、関係する全てのアイテム変数を、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて ZBDD を複製した
組合せ集合を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を指定する
ことはできない。

ZBDD ZBDD::operator<<(int)
自分自身のグラフに対して、関係する全てのアイテム変数を、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて ZBDD を複製した
オブジェクトを生成し、それを返す。実行結果において未定義の入力変数が必要に
なるような s を与えてはならない。必要な入力変数はあらかじめ宣言しておくこと。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは
何もしない。s に負の値を指定することはできない。

ZBDD ZBDD::operator>>(int)
自分自身のグラフに対して、関係する全てのアイテム変数を、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて ZBDD を複製した
オブジェクトを生成し、それを返す。実行結果において未定義の入力変数が必要に
なるような s を与えてはならない。必要な入力変数はあらかじめ宣言しておくこと。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは
何もしない。s に負の値を指定することはできない。

ZBDD ZBDD::OffSet(int var)
自分自身のグラフに対して、変数番号 var のアイテムを含まない組合せからなる
部分集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDD ZBDD::OnSet(int var)
自分自身のグラフに対して、変数番号 var のアイテムを含む組合せからなる
部分集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDD ZBDD::OnSet0(int var)
Onset(var)を実行した後、変数番号 var のアイテムを除去した集合を表す
ZBDD オブジェクトを生成し、それを返す。Onset(var).Change(var)と等価。
var がグラフの最上位の変数番号の場合は、1-エッジ が指しているサブグラフ
をそのまま返すことになる。記憶あふれの場合は、null を表すオブジェクトを
返す。自分自身が null だった場合は null を返す。

ZBDD ZBDD::Change(int)
自分自身のグラフに対して、変数番号 var のアイテムの有無を反転させた集合
を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合は、null を
表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDD Swap(int var1, int var2);
自分自身のグラフに対して、変数番号 var1 と var2 のアイテム変数を
入れ換えたときの論理関数を表す ZBDD オブジェクトを生成し、それを返す。引数は
level ではなく、変数番号で与えることに注意。記憶あふれの場合は、null を
表すオブジェクトを返す。自分自身が null のときは、null を返す。

ZBDD ZBDD::Restrict(ZBDD f)
自分自身の組合せ集合の要素となっている組合せの中で、f の中の少なくとも
１つの組合せを包含しているような組合せだけを抽出し、抽出した組合せ集合を表す
ZBDD オブジェクトを生成してそれを返す。記憶あふれの場合は、null を返す。
自分自身または引数が null の場合も null を返す

ZBDD ZBDD::Permit(ZBDD f)
自分自身の組合せ集合の要素となっている組合せの中で、f の中の少なくとも
１つの組合せに包含されているような組合せだけを抽出し、抽出した組合せ集合を表す
ZBDD オブジェクトを生成してそれを返す。記憶あふれの場合は、null を返す。
自分自身または引数が null の場合も null を返す

ZBDD ZBDD::PermitSym(int n)
自分自身の組合せ集合の要素となっている組合せの中で、アイテム個数が n 個以下の
組合せだけを抽出した組合せ集合を表す ZBDD オブジェクトを生成してそれを返す。
記憶あふれの場合は、null を返す。自分自身または引数が null の場合も null を返す

ZBDD ZBDD::Support(void)
自分自身の集合に現れるアイテムを抽出し、それらのアイテム１個ずつを要
素とする集合を表すオブジェクトを生成し、それを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。

int ZBDD::Top(void)
自分自身の組合せ集合に関係するアイテム変数のうち、最上位の順位を持つ
アイテムの変数番号を返す。null に対しては 0 を返す。

bddword ZBDD::GetID(void)
集合を一意に表現する 1-word の識別番号を返す。

bddword ZBDD::Size(void)
自分自身のグラフの節点数を返す。null に対しては 0 を返す。

bddword ZBDD::Card(void)
自分自身が表す集合の要素数(cardinality)を返す。null に対しては 0 を返す。

bddword ZBDD::Lit(void);
自分自身が表す集合中の総リテラル数（各組合せのアイテム個数の総和）を返す。
null に対しては 0 を返す。

void ZBDD::Export(FILE *strm = stdout)
ZBDD の内部データ構造を、strm で指定するファイルに出力する。

extern ZBDD ZBDD_LCM_A(char *fname, int th)
fname で指定する名前のファイルから FIMI ベンチマークフォーマットのトランザクション
データベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する頻出アイ
テム集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合が null を
返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

extern ZBDD ZBDD_LCM_C(char *fname, int th)
fname で指定する名前のファイルから FIMI ベンチマークフォーマットのトランザクション
データベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する飽和頻出
アイテム集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合が null
を返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

extern ZBDD ZBDD_LCM_M(char *fname, int th)
fname で指定する名前のファイルから FIMI ベンチマークフォーマットのトランザクション
データベースを読み込み、LCM アルゴリズムを用いて、閾値 th 回以上出現する極大頻出
アイテム集合を表す ZBDD オブジェクトを生成し、それを返す。記憶あふれの場合が null
を返す。ファイル読み込みに失敗した場合はエラーメッセージを出力する。

void ZBDD::PrintPla(void)
自分自身が表す集合を表形式（pla format）で標準出力に出力する。

void ZBDD::XPrint(void);
自分自身のグラフを、X-Window に描画する。

void ZBDD::XPrint0(void);
自分自身のグラフを、X-Window に描画する。(否定エッジなし)

void Print(void)
インデックスの値、最上位のリテラル番号、節点数の情報を標準出力に出力する。

****
クラス名: ZBDDV --- ZBDD の配列（組合せ集合の配列）を表すクラス
****
ヘッダーファイル名: "ZBDD.h"
ソースファイル名: ZBDD.cc
内部から呼び出しているクラス: BDD, BDDV, ZBDD

ZBDDV の配列を表すクラス。配列要素の番号は 0 から始まる整数である。
内部では BDDV と同様に出力選択変数を用いた二分木で処理している。
出力選択変数とユーザ変数の取り扱いは BDDV の場合と同様で、BDDV_Init()を
最初に実行する必要がある。ZBDDV は BDDV と異なり、、あらかじめ配列長を宣言する
必要はなく、要素にアクセスした瞬間にその要素の分だけのメモリが確保される。
ZBDDV 同士の演算で配列長が一致していない場合、足りない方の要素は 0 が仮定
される。未使用の要素を参照した場合も 0 が返される。

----------関連する定数値----------
extern const int BDDV_SysVarTop
extern const int BDDV_MaxLen
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar

----------関連する外部関数----------
extern ZBDDV operator&(const ZBDDV& fv, const ZBDDV& gv)
fv と gv の各配列要素同士の交わり(intersection)を表す ZBDDV オブジェクトを
生成し、それを返す。記憶あふれの場合は 長さ１の null を返す。自分自身が
null の場合には、長さ１の null を返す。

extern ZBDDV operator+(const ZBDDV& fv, const ZBDDV& gv)
fv と gv の各配列要素同士の結び(union)を表す ZBDDV オブジェクトを生成し、
それを返す。記憶あふれの場合は 長さ１の null を返す。自分自身が null の
場合には、長さ１の null を返す。

extern ZBDDV operator-(const ZBDDV& fv, const ZBDDV& gv)
fv の各配列要素 から gv の各配列要素を引いた差分集合を表す ZBDDV オブジェクトを
生成し、それを返す。記憶あふれの場合は 長さ１の null を返す。自分自身が
null の場合には、長さ１の null を返す。

extern int operator==(const ZBDDV& fv, const ZBDDV& gv)
fv と gv の各配列要素が同じ集合かどうかの真偽(1/0)を返す。

extern int operator!=(const ZBDDV& fv, const ZBDDV& gv)
fv と gv が各配列要素のうち少なくとも１つが異なる集合かどうかの真偽(1/0)を返す。

extern ZBDDV ZBDDV_Import(FILE *strm = stdin)
strm で指定するファイルから ZBDDV の構造を読み込み、ZBDDV オブジェクトを生成して、
それを返す。ファイルに文法誤りが合った場合等、異常終了時は null を返す。

（再掲）
extern void BDDV_Init(bddword init, bddword limit)
extern int BDD_NewVar(void)
extern int BDD_NewVarOfLev(int lev)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)
----------公開クラスメソッド----------
ZBDDV::ZBDDV(void)
基本 constructer。初期値として空集合を表す ZBDDV オブジェクトを生成する。

ZBDDV::ZBDDV(const ZBDDV& fv)
引数 fv を複製する constructer。

ZBDDV::ZBDDV(const ZBDD& f, int location = 0)
第 location 番目の配列要素が f で、それ以外の要素が 0 となっている ZBDDV
オブジェクトを生成する constructer。f が null だった場合は、location の値に
関わらず、0 番目の要素が null となっているオブジェクトを返す。

ZBDDV::~ZBDDV(void)
destructer。

ZBDDV& ZBDDV::operator=(const ZBDDV& fv)
自分自身に fv を代入し、fv を返す。

ZBDDV を複製した組合せ集合を生成し、それを返す。実行結果において未定義の入力
変数が必要になるような s を与えてはならない。必要な入力変数はあらかじめ宣言
しておくこと。記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が
null のときは何もしない。s に負の値を指定することはできない。

ZBDDV ZBDDV::operator>>(int s)
自分自身の各配列要素に対して、関係する全てのアイテム変数を、展開順位
(level)が s ずつ小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて
ZBDDV を複製した組合せ集合を生成し、それを返す。実行結果において未定義の入力
変数が必要になるような s を与えてはならない。必要な入力変数はあらかじめ宣言
しておくこと。記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が
null のときは何もしない。s に負の値を指定することはできない。

ZBDDV ZBDDV::OffSet(int var)
自分自身の各配列要素について、変数番号 var のアイテムを含まない組合せからなる
部分集合を表す ZBDDV オブジェクトを生成し、それを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDDV ZBDDV::OnSet(int var)
自分自身の各配列要素に対して、変数番号 var のアイテムを含む組合せからなる
部分集合を表す ZBDDV オブジェクトを生成し、それを返す。記憶あふれの場合は、
null を表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDDV ZBDDV::OnSet0(int var)
Onset(var)を実行した後、変数番号 var のアイテムを除去した集合のベクトルを
表す ZBDDV オブジェクトを生成し、それを返す。Onset(var).Change(var)と等価。
var がグラフの最上位の変数番号の場合は、1-エッジ が指しているサブグラフ
をそのまま返すことになる。記憶あふれの場合は、null を表すオブジェクトを
返す。自分自身が null だった場合は null を返す。

ZBDDV ZBDDV::Change(int var)
自分自身のグラフに対して、変数番号 var のアイテムの有無を反転させた集合
を表す ZBDDV オブジェクトを生成し、それを返す。記憶あふれの場合は、null を
表すオブジェクトを返す。自分自身が null だった場合は null を返す。

ZBDDV ZBDDV::Swap(int var1, int var2)

ZBDDV ZBDDV::operator&=(const ZBDDV& fv)
自分自身と fv との各配列要素同士の交わり(intersection)を求め、自分自身に
代入する。記憶あふれの場合は 長さ１の null を代入する。自分自身または
引数に null が含まれていた場合も、長さ１の null となる。

ZBDDV ZBDDV::operator+=(const ZBDDV& fv)
自分自身と fv との各配列要素同士の結び(union)を求め、自分自身に代入する。
記憶あふれの場合は 長さ１の null を代入する。自分自身または引数に null が
含まれていた場合には、長さ１の null をとなる。

ZBDDV ZBDDV::operator-=(const ZBDDV& fv)
自分自身の各配列要素から fv の各配列要素を引いた差分集合を求め、自分自身
に代入する。記憶あふれの場合は 長さ１の null を代入する。自分自身または引
数に null が含まれていた場合には、長さ１の null となる。

ZBDDV ZBDDV::operator<<=(int s)
自分自身の各配列要素に対して、関係する全てのアイテム変数を、展開順位
(level)が s ずつ大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて
ZBDDV を複製した組合せ集合を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を指定する
ことはできない。

ZBDDV ZBDDV::operator>>=(int s)
自分自身の各配列要素に対して、関係する全てのアイテム変数を、展開順位
(level)が s ずつ小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて
ZBDDV を複製した組合せ集合を、自分自身に代入する。また演算結果を関数値として返す。
実行結果において未定義の入力変数が必要になるような s を与えてはならない。
必要な入力変数はあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を指定する
ことはできない。

ZBDDV ZBDDV::operator<<(int s)
自分自身の各配列要素に対して、関係する全てのアイテム変数を、展開順位
(level)が s ずつ大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて

自分自身のグラフに対して、変数番号 var1 と var2 のアイテム変数を
入れ換えたときの論理関数を表す ZBDDV オブジェクトを生成し、それを返す。引数は
level ではなく、変数番号で与えることに注意。記憶あふれの場合は、null を
表すオブジェクトを返す。自分自身が null のときは、null を返す。

int ZBDDV::Top(void)
自分自身の配列要素の中で、関係するアイテム変数のうち最上位の順位を持つ
アイテムの変数番号を返す。null に対しては 0 を返す。

ZBDDV ZBDDV::Mask(int start, int length = 1)
自分自身の第 start 番目から 第(start+length)番目までの配列要素を残し、
その他の配列要素を 0（空集合）とした ZBDDV オブジェクトを生成し、それを返す。
記憶あふれの場合は 長さ１の null を返す。自分自身が null の場合には、長さ
１の null を返す。

ZBDD ZBDDV::GetZBDD(int index)
自分自身の第 index 番目の配列要素の ZBDD オブジェクトを生成し、それを返す。

ZBDD ZBDDV::GetMetaZBDD(void)
自分自身の内部構造を表す ZBDD オブジェクト（出力選択変数を含む）を返す。

int ZBDDV::Last(void)
自分自身の（意味のある）配列要素の中の、最大の要素番号を返す。null に対
しては 0 を返す

bddword ZBDDV::Size(void);
自分自身のグラフの節点数を返す（出力選択変数に関する節点は含まない）。
null に対しては 0 を返す。

void ZBDDV::Print(void)
インデックスの値、最上位のリテラル番号、ノード数の情報を標準出力に出力する。

void ZBDDV::Export(FILE *strm = stdout)
ZBDD の内部データ構造を、strm で指定するファイルに出力する。

int ZBDDV::PrintPla(void)

自分自身が表す集合を表形式（pla format）で標準出力に出力する。関数の値は、
通常 0 を返す。記憶あふれの場合は、出力を中断し 1 を返す。自分自身が null の
場合には、何も出力せず 1 を返す。

void ZBDDV::XPrint(void)
自分自身のグラフを、X-Window に描画する。

void ZBDDV::XPrint0(void);
自分自身のグラフを、X-Window に描画する。（否定エッジなし）

****
クラス名: Ctol　---整数値組合せ集合（整係数ユネイト論理式）を表すクラス
****
ヘッダーファイル名: "Ctol.h"
ソースファイル名: Ctol.cc
内部から呼び出しているクラス: ZBDD, ZBDDV

整数値組合せ集合を表すクラス。整数値組合せ集合は、複数個のアイテムの有無の
組合せを要素とする集合であって、なおかつ各要素が整数の値を持つことができる。
内部データは整数値を符号化し、ZBDD を用いて表現されている。(-2)進数を採用して
いるため､負数も正数と同様に表現される｡整数値の桁数の上限は約 100 万となっており、
実質的には青天井である。（ただし桁数が大きくなると処理速度は低下する）。
記憶あふれの場合は、Ctol_Null()を返す（以下、null と呼ぶ）。

----------関連する定数値----------
extern const int BDDV_SysVarTop
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar

----------関連する外部関数----------
extern int operator==(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b が同じ整数値組合せ集合を表すならば 1、
そうでなければ 0 を返す。Ctol_EQ() とは意味が違うので注意。

extern int operator!=(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b が同じ整数値論理関数を表すならば 0、
そうでなければ 1 を返す。Ctol_NE() とは意味が違うので注意。

extern Ctol operator+(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b の算術和（対応する要素同士の整数値の加算）
を表す Ctol オブジェクトを新しく生成し、それを値として返す。記憶あふれの
場合は null を返す。引数に null が含まれる場合は null を返す。桁あふれの
場合はエラー終了する。

extern Ctol operator-(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b の算術差（対応する要素同士の整数値の減算）
を表す Ctol オブジェクトを新しく生成し、それを値として返す。記憶あふれの
場合は null を返す。引数に null が含まれる場合は null を返す。桁あふれの
場合はエラー終了する。

extern Ctol operator*(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b の算術積を表す Ctol オブジェクトを新しく生成
し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。桁あふれの場合はエラー終了する。この算術積演算は、
a の要素と b の要素を１つずつ取り出してできる全ての組合せについて乗算を行い、
それらの総和を求めるものである。同じアイテム変数を２乗してもべき乗には
ならないこと（v × v = v）を除けば、整数係数の多項式の乗算とほぼ同様の演算で
ある。例えば、(x + 2 y)(x + 3 y) = x + 6 y + 5 x y となる。

extern Ctol operator/(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b の整数除算の商を表す Ctol オブジェクトを新し
く生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null
が含まれる場合は null を返す。結果が負で割り切れないときは、絶対値の小さ
い方に切り捨てる（例：10/(-3)= -3 ）。除数 b が多項式（複数の組合せ要素を持つ
集合）の場合は、b の中から要素を１つずつ取り出して除算を行ったときに、b のどの
要素で割っても商に必ず含まれる組合せ要素の集合を、b 全体で割ったときの商と
定義する。このとき、各組合せ要素の整数値は、部分商に出現する組合せ要素の
整数値の中で絶対値が最小であるものを全体の商の整数値とする。
例えば、(20 x + 12 x y + 10 y + 8) / (3 x + 2) の場合、
(20 x + 12 x y + 10 y + 8)/(3 x) = 6 + 4 y,
(20 x + 12 x y + 10 y + 8)/2 = 10 x + 6 x y + 5 y + 4
なので、全体の商は 4 y + 4 となる。

extern Ctol operator%(const Ctol& a, const Ctol& b)
２つの Ctol オブジェクト a, b の整数除算の剰余を表す Ctol オブジェクトを新しく
生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含ま
れる場合は null を返す。負数やアイテム変数が含まれている場合でも、
(a % b) = a - (b * (a / b))　を満たすように計算する。(例) -10 % 3 = -1。

extern Ctol Ctol_Null(void)
エラーや記憶あふれを表す Ctol オブジェクト(null)を生成し、それを値として返す。
実際の内容は ZBDD(-1)である。

extern Ctol Ctol_ITE(Ctol a, Ctol b, Ctol c)
if-then-else の演算。すなわち､a に含まれる組合せ要素に対しては b のデータを複製し、
a に含まれない組合せ要素に対しては、c のデータを複製したような Ctol オブジェクトを
新しく生成し、それを返す。記憶あふれの場合は null を返す。引数に null が含まれる場
合は null を返す。a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

extern Ctol Ctol_EQ(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a と b で整数値が
等しい組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。operator == とは意味が違うので注意。記憶あふれの場合は null を返す。引数に
null が含まれる場合は null を返す。

extern Ctol Ctol_NE(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a と b で整数値が
等しくない組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。operator != とは意味が違うので注意。記憶あふれの場合は null を返す。引数に
null が含まれる場合は null を返す。

extern Ctol Ctol_GT(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値が b よりも
大きい組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_GE(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値が b よりも
小さくない組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_LT(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値が b よりも
小さい組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_LE(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値が b よりも
大きくない組合せ要素だけを取り出した集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_Max(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値と b の値の
大きい方を選ぶ組合せ集合の Ctol オブジェクトを新しく生成し、それを返す。
正負の符号が異なる場合は正の数値が選ばれる。負数同士であれば絶対値の小さい方が
選ばれる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_Min(Ctol a, Ctol b)
a と b の少なくとも一方で 0 以外の整数値を持つ組合せ要素について、a の値と b の値の
小さい方を選ぶ組合せ集合の Ctol オブジェクトを新しく生成し、それを返す。
正負の符号が異なる場合は負の数値が選ばれる。負数同士であれば絶対値の大きい方が
選ばれる。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_atoi(char* s)
s の指す数字文字列を数値に変換し、定数項のみを持つ Ctol オブジェクトを新しく
生成し、それを値として返す。通常は 10 進数として変換するが、文字列の先頭

が ``0X'' または ``0x'' で始まる場合には 16 進数、 ``0B'' または ``0b'' で始まる場合は２進数で変換する。それ以外の不適当な文字を含む文字列に対する動作は保証しない。桁数は約 100 万ビットまで事実上無制限に処理できる。

extern Ctol Ctol_Intsec(Ctol a, Ctol b)
２つの Ctol オブジェクト a, b に関して、(-2)進数の各ビットごとの集合積(intersection)を表す Ctol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_Union(Ctol a, Ctol b)
２つの Ctol オブジェクト a, b に関して、(-2)進数の各ビットごとの集合和(union)を表す Ctol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol Ctol_Diff(Ctol a, Ctol b)
２つの Ctol オブジェクト a, b に関して、(-2)進数の各ビットごとの集合差(Difference)を表す Ctol オブジェクトを新しく生成し、それを値として返す。記憶あふれの場合は null を返す。引数に null が含まれる場合は null を返す。

extern Ctol_Meet(Ctol a, Ctol b)
a と b の Meet 演算 (Knuth 本 4 巻 1 分冊 141 頁: 演習問題 203 参照。ただし、整数値の重みつき集合に拡張) により得られる集合を表す Ctol オブジェクトを生成し、それを返す記憶あふれの場合は、null を表すオブジェクトを返す。引数に null を与えた場合には null を返す。この Meet 演算は、a の要素と b の要素を 1 つずつ取り出してできる全ての組合せについて共通のアイテム組合せを取り出し、それらの総和を求めるものである。これは、整数係数の多項式の乗算を行った結果の式から、1 次の変数を消去して、2 次の変数を 1 次に書き換えて残したものと考えてよい。例えば、Meet(x + 2 x y + 3 y, x + y + 1) = x + 1 + 1 + 2 x + 2 y + 2 + 3 + 3 y + 3 = 3 x + 5 y + 10 となる。

（再掲）
extern void BDDV_Init(bddword init, bddword limit)
extern int BDD_NewVar(void)
extern int BDD_NewVarOfLev(int lev)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)

----------公開クラスメソッド----------
Ctol::Ctol(void)
基本 constructer。初期値として空集合（全ての要素の値が 0）を表すオブジェクトを生成する。

Ctol::Ctol(const Ctol& a)
Ctol オブジェクトのコピーを生成する constructer。

Ctol::Ctol(const ZBDD& f)
自分自身の内部データの ZBDD オブジェクト f を与える constructor。

Ctol::Ctol(int n)
値 n の定数項（アイテム変数に依存しない組合せ）のみからなる組合せ集合を表す Ctol オブジェクトを生成する constructer。

Ctol::~Ctol(void)
destructer。

Ctol& Ctol::operator=(const Ctol& a)
a を自分自身に代入し、それを値として返す。

int Ctol::Top(void)
自分自身の組合せ集合に関する変数の中で、最高の順位(level)を持つ変数の番号(VarID)を返す。level の値そのものを返すわけではないので注意。変数に依存しない（0 または定数 1）のときは 0 を返す。null のときは 0 を返す。Ctol クラスでは整数値を表すために特殊な変数を使用しており、その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。

int Ctol::TopItem(void)
自分自身の組合せ集合に関する変数の中で、整数値を表すための特殊変数を除いて最高の順位(level)を持つアイテム変数の番号(VarID)を返す。level の値そのものを返すわけではないので注意。整数値の定数項のみを含む組み合わせ集合に対しては 0 を返す。null のときは 0 を返す。

int Ctol::TopDigit(void)
自分自身が保持している整数値について、(-2)進数で表現したときの最上位桁の桁番号（桁数-1）を返す。空集合に対しては 0 を返す。null のときは 0 を返す。

int IsBool(void)
自分自身がブール集合（整数値が 0 または 1）であるかどうかの真偽(1/0)を返す。自分自身が null だった場合は 1 を返す。

int IsConst(void)
自分自身が定数項（アイテム変数に依存しない項）のみからなるかどうかの真偽(1/0)を返す。自分自身が null だった場合は 1 を返す。

Ctol Ctol::AffixVar(int var)
自分自身が表す組合せ集合に対して、変数番号 var のアイテム変数を各要素に付加したときの組合せ集合を表す Ctol オブジェクトを新しく生成し、それを返す。変数 v を元々含んでいた組合せ要素については、この演算による変化はない。Ctol クラスでは整数値を表すために特殊な変数を使用しており、その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。AffixVar 演算では、特殊変数でもアイテム変数と同様に処理される。

Ctol Ctol::Factor0(int var)
自分自身を、変数番号 var の変数で割ったときの剰余を返す。すなわち変数 var を含まない組合せ要素をすべて取りだした Ctol オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が null のときには null を返す。Ctol クラスでは整数値を表すために特殊な変数を使用しており、その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。Factor0 演算では、特殊変数でもアイテム変数と同様に処理される。

Ctol Ctol::Factor1(int var)
自分自身を、変数番号 var の変数で割ったときの商を返す。すなわち変数 var を含む組合せ要素をすべて取りだし、その各要素から該当変数を取り除いた Ctol オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が null のときには null を返す。Ctol クラスでは整数値を表すために特殊な変数を使用しており、その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。Factor1 演算では、特殊変数でもアイテム変数と同様に処理される。

Ctol Ctol::FilterThen(Ctol a)
自分自身のデータのうち、a に含まれる組合せ要素に関するものだけを抽出複製した Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。条件を表す組合せ集合 a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

Ctol Ctol::FilterElse(Ctol a)
自分自身のデータのうち、a に含まれない組合せ要素に関するものだけを抽出複製した Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。条件を表す組合せ集合 a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

Ctol Ctol::FilterRestrict(Ctol a)
自分自身のデータのうち、a の組合せ要素の中の少なくとも 1 つを包含するような組合せだけを抽出複製した Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。条件を表す組合せ集合 a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

Ctol Ctol::FilterPermit(Ctol a)
自分自身のデータのうち、a の組合せ要素の中の少なくとも 1 つに包含されるような組合せだけを抽出複製した Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。条件を表す組合せ集合 a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

Ctol Ctol::FilterPermitSym(int n)
自分自身のデータのうち、アイテム個数が n 個以下の組合せだけを抽出複製した Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。条件を表す組合せ集合 a に関しては、0 以外の整数値は全て 1 と同じ意味を持つ。

Ctol Ctol::Support(void)
自分自身の集合に現れるアイテムを抽出し、それらのアイテム 1 個ずつを要素とする集合を表す Ctol オブジェクトを生成し、それを返す。記憶あふれの

場合は、null を表すオブジェクトを返す。

Ctol Ctol::NonZero(void)
自分自身に含まれる組合せ要素をすべて抽出した Ctol オブジェクトを新しく生成し、
それを返す。0 以外の整数値を持つ組合せ要素を全て 1 に正規化しブール組合せ
集合を生成する働きを持つ。記憶あふれの場合は null を返す。

Ctol Ctol::Digit(int d)
自分自身の第 d 桁目の組合せ集合を表す Ctol オブジェクトを新しく生成し、
それを返す。演算結果はブール組合せ集合（整数値は 0 または 1）となる。
自分自身が null の場合は null を返す。d が負の場合はエラー終了する。
記憶あふれの場合は null を返す。

Ctol Ctol::EQ_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
その整数値が等しい組合せのみを抽出した Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。

Ctol Ctol::NE_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
その整数値が等しくない組合せのみを抽出した Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。

Ctol Ctol::GT_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
整数値が a の定数値より大きい組合せのみを抽出した Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。

Ctol Ctol::GE_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
整数値が a の定数値より小さくない組合せのみを抽出した Ctol オブジェクトを新しく
生成し、それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1
または 0）となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は
null を返す。

Ctol Ctol::LT_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
整数値が a の定数値より小さい組合せのみを抽出した Ctol オブジェクトを新しく生成し、
それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1 または 0）
となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は null を返す。

Ctol Ctol::LE_Const(Ctol a)
自分自身の各組合せ要素と a の定数項（アイテム変数に依存しない要素）を比較し、
整数値が a の定数値より大きくない組合せのみを抽出した Ctol オブジェクトを新しく
生成し、それを返す。演算結果は比較条件の成否を表すブール組合せ集合（整数値は 1
または 0）となる。記憶あふれの場合は null を返す。引数や自分自身が null の場合は
null を返す。

Ctol Ctol::MaxVal(void)
自分自身が含む組合せ要素の中で最大の整数値を表す、定数項の Ctol オブジェクトを
新しく生成し、それを値として返す。記憶あふれの場合は null を返す。
自分自身が null の場合は null を返す。

Ctol Ctol::MinVal(void)
自分自身が含む組合せ要素の中で最小の整数値を表す、定数項の Ctol オブジェクトを
新しく生成し、それを値として返す。記憶あふれの場合は null を返す。
自分自身が null の場合は null を返す。

Ctol Ctol::TotalVal(void)
自分自身の各組合せ要素の整数値の総和を表す、定数項の Ctol オブジェクトを
新しく生成し、それを値として返す。記憶あふれの場合は null を返す。
自分自身が null の場合は null を返す。

ZBDD Ctol::GetMetaZBDD(void)
自分自身の内部データの ZBDD オブジェクトを複製して、それを返す。

Ctol Ctol::CountTerms(void)
自分自身が含む組合せ要素の総数を表す、定数項の Ctol オブジェクトを新しく生成し、
それを値として返す。記憶あふれの場合は null を返す。自分自身が null の場合は
null を返す。

Ctol Ctol::Abs(void)
自分自身の組合せ集合の整数値を絶対値に書き換えた Ctol オブジェクトを新しく
生成し、それを返す。記憶あふれの場合は null を返す。自分自身が null の場合は
null を返す。

Ctol Ctol::Sign(void)
自分自身の組合せ集合の各要素の符号を調べ、正なら 1, 負なら-1 の整数値に書き換えた
Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。
自分自身が null の場合は null を返す。

Ctol Ctol::operator -(void)
自分自身の組合せ集合の整数値の正負を反転させた Ctol オブジェクトを新しく生成し、
それを返す。記憶あふれの場合は null を返す。自分自身が null の場合は null を返す。

Ctol Ctol::TimesSysVar(int var)
自分自身に変数番号 var の特殊変数を掛け算して得られる Ctol オブジェクトを新しく
生成し、それを返す。記憶あふれの場合は null を返す。自分自身が null の場合は
null を返す。Ctol クラスでは整数値を表すために特殊な変数を使用しており、
その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。範囲外の
var を与えた場合はエラー終了する。本演算は(-2)の 2 のべき乗を掛け算するという
意味を持つ。

Ctol Ctol::DivBySysVar(int var)
自分自身を変数番号 var の特殊変数で割り算して得られる Ctol オブジェクトを新しく
生成し、それを返す。記憶あふれの場合は null を返す。自分自身が null の場合は
null を返す。Ctol クラスでは整数値を表すために特殊な変数を使用しており、
その変数番号は 1～BDDV_SysVarTop (=通常 20) までの範囲となっている。範囲外の
var を与えた場合はエラー終了する。本演算は(-2)の 2 のべき乗で割り算するという
意味を持つ。

Ctol Ctol::ShiftDigit(int power)
自分自身に含まれる各組合せ要素の整数値を、桁数 power だけシフトさせて得られる
Ctol オブジェクトを新しく生成し、それを返す。記憶あふれの場合は null を返す。
自分自身が null の場合は null を返す。power が正の場合は左シフト (-2 のべき乗倍)、
負の場合は右シフト (-2 のべき乗での割り算)、0 の場合は変化なし。

Ctol Ctol::operator+=(const Ctol& a)
自分自身と a の算術和を自分自身に代入し、それを値として返す。記憶あ
ふれの場合は null を返す。自分自身または引数が null の場合は null を返す。
算術和の詳細は operator+を参照。

Ctol Ctol::operator-=(const Ctol& a)
自分自身と a の算術差を自分自身に代入し、それを値として返す。記憶あ
ふれの場合は null を返す。自分自身または引数が null の場合は null を返す。
算術差の詳細は operator-を参照。

Ctol Ctol::operator*=(const Ctol& a)
自分自身と a の算術積を自分自身に代入し、それを値として返す。記憶あ
ふれの場合は null を返す。自分自身または引数が null の場合は null を返す。
算術積の詳細は operator*を参照。

Ctol Ctol::operator/=(const Ctol& a)
自分自身と a の整数除算の商を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返
す。整数除算の詳細は operator/を参照。

Ctol Ctol::operator%=(const Ctol& a)
自分自身と a の整数除算の剰余を自分自身に代入し、それを値として返す。
記憶あふれの場合は null を返す。自分自身または引数が null の場合は null を返
す。整数除算の詳細は operator/, operator%を参照。

bddword Ctol::Size()
自分自身のグラフの節点数を返す。null の場合は 0 を返す。

int Ctol::GetInt(void)
自分自身の定数項（アイテム変数に依存しない要素）の値を返す。整数値が
32 ビットに収まらない場合は、下位 32 ビットのみを有効とする。null の
場合は 0 を返す。

int Ctol::StrNum10(char* s)

自分自身の定数項（アイテム変数に依存しない要素）の値を表す 10 進数文字列を
生成し、s から始まる領域に書き込む。s には事前に十分な領域（最低 10 バイト）
を確保しておかなければならない。関数の値は通常 0 を返すが、記憶あふれの場
合や、null のときは 1 を返す。このとき文字列には"0"が入る。

int Ctol::StrNum16(char* s)
自分自身の定数項（アイテム変数に依存しない要素）の値を表す 16 進数文字列を
生成し、s から始まる領域に書き込む。s には事前に十分な領域（最低 10 バイト）
を確保しておかなければならない。関数の値は通常 0 を返すが、記憶あふれの場
合や、null のときは 1 を返す。このとき文字列には"0"が入る。

void Ctol::PutForm(void)
自分自身を数式形式のテキスト列で表現し、標準出力に出力する。

void Ctol::Print(void)
自分自身の内部データ情報を標準出力に出力する。

void Ctol::XPrint(void)
自分自身の内部データ情報を X-Window に描画する。

void Ctol::XPrint0(void)
自分自身の内部データ情報を X-Window に描画する。（否定枝を使わない状態で表示する。）

****
クラス名: SOP　---正負のリテラルからなる積和形論理式を表現するクラス
****
ヘッダーファイル名: "SOP.h"
ソースファイル名: SOP.cc
内部から呼び出しているクラス: BDD, ZBDD

ZBDD を用いて、積和形論理式を表したクラスである。積和形論理式は積項の
和集合であり、積項はリテラルの積集合である。リテラルはそれぞれの入力
変数について、正リテラルと負リテラルの２種類を使用する。１つの積項に、
同じ入力変数の正負両方のリテラルが同時に含まれることはない。SOP_NewVar()を
用いて変数を宣言することにより、正リテラルの変数番号(VarID)は(2 k)番,
負リテラル(2 k - 1)番が割り当てられる。BDD と SOP を同時に対応させて使う場合は、
BDD の入力変数は SOP の正リテラルの変数番号(VarID)を使用する。

（使用例）
```
int var1 = SOP_NewVar();
int var2 = SOP_NewVar();
int var3 = SOP_NewVar();
int var4 = SOP_NewVar();
SOP f1 = SOP(1).And1(var1);
SOP f2 = SOP(1).And0(var2);
SOP f3 = f1 + f2;
SOP f4 = f.And1(var3) + f.And0(var4);
f3.Print();
f4.Print();
```

----------関連する定数値----------
```
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar
```

----------関連する外部関数----------
extern int SOP_NewVar(void)
新しいリテラル（正負１組）を生成し、正リテラルの変数番号(通称 VarID)を返す。
負リテラルの VarID は正リテラルよりも 1 小さい数値となる。VarID は 1 から
始まる整数で、SOP_NewVar()または SOP_NewVarOfLev()を 1 回実行するごとに
2 ずつ大きな値が返る。生成したリテラルの ZBDD 展開順位(通称 level)は、VarID と
同じ値となる。リテラルの個数が最大値 BDD_MaxVar を超えるとエラーを出力して
異常終了する。なお、最初に BDDV_Init()で初期化した場合（SOPV クラスを扱う場合
には、最初にシステム用に変数が使われるので、VarID は（BDDV_SysVarTop + 1)から
開始し、順に 1 ずつ大きな値となる。

extern int SOP_NewVarOfLev(int lev)
新しいリテラル（正負１組）を生成し、正リテラルの変数番号(通称 VarID)を返す。
負リテラルの VarID は正リテラルよりも 1 小さい数値となる。VarID は 1 から
始まる整数で、SOP_NewVar()または SOP_NewVarOfLev()を 1 回実行するごとに
2 ずつ大きな値が返る。生成するリテラルの BDD 展開順位(通称 level)は、
引数 lev で指定した値となる。実行時に順位 lev のリテラルがすでに存在していた
場合は、lev 以上の変数を 2 つずつ上にずらして（level を 2 ずつ増加させて)、空いた
ところに新しいリテラルを挿入する。引数 lev は 2 以上かつ「関数実行直前のリテラルの
個数+2」以下の偶数でなければならない。そうでなければエラーを出力して異常終了する。

extern SOP operator&(const SOP& f, const SOP& g)
f と g の交わり(intersection)を表す SOP オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOP operator+(const SOP& f, const SOP& g)
f と g の結び(union)を表す SOP オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOP operator-(const SOP& f, const SOP& g)
f から g を引いた差分集合を表す SOP オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

extern int operator==(const SOP& f, const SOP& g)
f と g が同じ集合かどうかの真偽(1/0)を返す。

extern int operator!=(const SOP& f, const SOP& g)
f と g が異なる集合かどうかの真偽(1/0)を返す。

extern SOP operator*(const SOP& f, const SOP& g)
f と g の直積（算術乗算）を表す SOP オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOP operator/(const SOP& f, const SOP& g)
f を g で割った商（Weak division）を表す SOP オブジェクトを生成し、
それを返す。記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOP operator%(const SOP& f, const SOP& g)
f を g で割った余り（Weak division の剰余）を表す SOP オブジェクトを生成し、
それを返す。記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOP SOP_ISOP(BDD f)
BDD で与えられた論理関数 f に対して、その非冗長積和形を表す SOP オブジェクトを
生成し、それを返す。記憶あふれの場合や引数に null が含まれているときには null
を返す。

extern SOP SOP_ISOP(BDD on, BDD dc)
オンセット on、ドントケアセット dc の組によって表される論理関数に対して、
その非冗長積和形を表す SOP オブジェクトを生成し、それを返す。記憶あふれの
場合や引数に null が含まれているときには null を返す。

（再掲）
```
void BDD_Init(bddword init, bddword limit)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)
```

----------公開クラスメソッド----------
SOP::SOP(void)
基本 constructer。初期値として空集合（恒偽式）を表す SOP オブジェクトを生成する。

SOP::SOP(int val)
定数式を作り出す constructer。val == 0 ならば恒偽式、val > 0 ならば恒真式、
val < 0 ならば null を表す SOP オブジェクトを生成する。

SOP::SOP(const SOP& f)
引数 f を複製する constructer。

SOP::SOP(const ZBDD& zbdd)
内部表現の ZBDD 表現 zbdd を引数として、それを複製する constructer。

SOP::~SOP(void)
destructer。

SOP& SOP::operator=(const SOP& f)
自分自身に f を代入し、f を返す。

SOP SOP::operator&=(const SOP& f)
自分自身と f との交わり(intersection)を求め、自分自身に代入する。
記憶あふれの場合は null を代入する。自分自身や引数が null のときには null となる。

SOP SOP::operator+=(const SOP& f)
自分自身と f との結び(union)を求め、自分自身に代入する。記憶あふれの場合は
null を代入する。自分自身や引数が null のときには null となる。

SOP SOP::operator-=(const SOP& f)
自分自身から f を引いた差分集合を求め、自分自身に代入する。記憶あふれの
場合は null を代入する。自分自身や引数が null のときには null となる。

SOP SOP::operator*=(const SOP& f)
自分自身と f の直積（算術乗算）を求め、自分自身に代入する。記憶あふ
れの場合は null を代入する。自分自身や引数が null のときには null となる。

SOP SOP::operator/=(const SOP& f)
自分自身を f で割った商（weak division）を求め、自分自身に代入する。記憶
あふれの場合は null を代入する。自分自身や引数が null のときには null となる。

SOP SOP::operator%=(const SOP& f)
自分自身を f で割った余り（weak division の剰余）を求め、自分自身に代入す
る。記憶あふれの場合は null を代入する。自分自身や引数が null のときには

SOP SOP::operator<<=(int)
自分自身のグラフに対して、関係する全てのリテラルを、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
積項集合を、自分自身に代入する。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOP SOP::operator>>=(int s)
自分自身のグラフに対して、関係する全てのリテラルを、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した
積項集合を、自分自身に代入する。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOP SOP::operator<<(int s)
自分自身のグラフに対して、関係する全てのリテラルを、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した SOP
オブジェクトを生成し、それを返す。s は偶数でなければならない。実行結果に
おいて未定義のリテラルが必要になるような s を与えてはならない。必要な
リテラルはあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を
指定することはできない。

SOP SOP::operator>>(int s)
自分自身のグラフに対して、関係する全てのリテラルを、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した SOP
オブジェクトを生成し、それを返す。s は偶数でなければならない。実行結果に
おいて未定義のリテラルが必要になるような s を与えてはならない。必要な
リテラルはあらかじめ宣言しておくこと。記憶あふれの場合は、null を表す
オブジェクトを返す。自分自身が null のときは何もしない。s に負の値を
指定することはできない。

SOP SOP::And0(int var)
自分自身の各積項に、変数番号 var の負リテラルを追加した積項集合の SOP
オブジェクトを生成し、それを返す。ただし、同じ番号の正リテラルをすでに
含む項は消される。記憶あふれの場合や自分自身が null のときには null を返す。

SOP SOP::And1(int var)
自分自身の各積項に、変数番号 var の正リテラルを追加した積項集合の SOP
オブジェクトを生成し、それを返す。ただし、同じ番号の負リテラルをすでに
含む項は消される。記憶あふれの場合や自分自身が null のときには null を返す。

SOP SOP::Factor0(int var)
自分自身を、変数番号 var の負リテラルで割ったときの商を返す。すなわち、
該当リテラルを含む項をすべて取りだし、その各項から該当リテラルを取り除
いた SOP オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が
null のときには null を返す。

SOP SOP::Factor1(int var)
自分自身を、変数番号 var の正リテラルで割ったときの商を返す。すなわち、
該当リテラルを含む項をすべて取りだし、その各項から該当リテラルを取り除
いた SOP オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が
null のときには null を返す。
SOP SOP::FactorD(int var)
自分自身の積項のうち、変数番号 var の正・負リテラルをどちらも含まない
項からなる SOP オブジェクトを生成し、それを返す。記憶あふれの場合や
自分自身が null のときには null を返す。

int SOP::Top(void)
自分自身のグラフが含んでいるリテラルの中で、最上位の展開順位を持つリテラル
（正リテラル）の変数番号を返す。必ず偶数になる。null に対しては 0 を返す。

SOP SOP::Swap(int var1, int var2)
自分自身のグラフに対して、変数番号 var1 と var2 のリテラルを入れ換えた
ときの積項集合を表す SOP オブジェクトを生成し、それを返す。var1, var2 は
正リテラルの番号（偶数）を指定するだけで、負リテラルも同時に入れ替えが実行
される。引数は level ではなく、変数番号で与えることに注意。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは、null を
返す。

bddword SOP::Size(void)
自分自身のグラフの節点数を返す。null に対しては 0 を返す。

bddword SOP::Cube(void)
自分自身の積項数を返す。null に対しては 0 を返す。

bddword SOP::Lit(void)
自分自身の総リテラル数(各積項のリテラル数の総和)を返す。null に対しては 0 を返す。

int SOP::IsPolyCube(void)
自分自身が複数個の積項を持つ場合（多項式）には 1、そうでなければ 0
を返す。null に対しては 0 を返す。記憶あふれの場合は 0 を返す。

int SOP::IsPolyLit(void)
自分自身が複数個のリテラルを持つ場合には 1、そうでなければ 0 を返す。
null に対しては 0 を返す。記憶あふれの場合は 0 を返す。

SOP SOP::Divisor(void)
自分自身を割るときの除数候補の１つを表す SOP オブジェクトを生成し、それを返
す。0 に対しては 0 を返す。1 に対しては 1 を返す。単項式に対しては 1 を返す。
多項式で同じリテラルが２度現れない場合には、自分自身のコピーを返す。
同じリテラルが２度以上現れる場合には、商にそのリテラルが含まれる
ような除数を返す。得られた除数には、同じリテラルは２度現れない。null を
与えた場合や記憶あふれの場合は null を返す。

SOP SOP::Implicants(BDD f)
自分自身の積項のうち、f で与えた論理関数の内項（f=1 となる部分のみをカバー
している積項）を抽出し、その集合を表す SOP オブジェクトを生成して返す。記憶
あふれの場合は、null を表すオブジェクトを返す。

SOP SOP::Support(void)
自分自身の集合に現れる正または負のリテラルを抽出し、抽出されたリテラルの
正リテラル１個ずつを要素とする集合を表す SOP オブジェクトを生成し、それを返す。
記憶あふれの場合は、null を表すオブジェクトを返す。

void SOP::Print(void)
インデックスの値、最上位のリテラル番号、ノード数、積項数、リテラル数の
情報を標準出力に出力する。

int SOP::PrintPla(void)
自分自身が表す集合を表形式（pla format）で標準出力に出力する。

ZBDD SOP::GetZBDD(void)
内部表現の ZBDD を複製したオブジェクトを生成して、それを返す。

BDD SOP::GetBDD(void)
自分自身の積項集合が表す論理関数の BDD オブジェクトを生成し、それを返す。
記憶あふれの場合や自分自身が null のときには null を返す。

SOP SOP::InvISOP(void)
自分自身の積項集合が表す論理関数の否定関数を求め、その非冗長積和形を
表す SOP オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が
null のときには null を返す。

****
クラス名: SOPV　--- SOP の配列（積和形論理式の配列）を表すクラス
****
ヘッダーファイル名: "SOP.h"
ソースファイル名: SOP.cc
内部から呼び出しているクラス: BDD, ZBDD, SOP

SOP の配列を表すクラス。配列要素の番号は 0 から始まる整数である。ZBDDV と
同様に、内部では出力選択変数を用いた二分木で処理している。あらかじめ
配列長を宣言する必要はなく、要素にアクセスした瞬間にその要素の分だけの
メモリが確保される。SOPV 同士の演算で配列長が一致していない場合、足りない方
の要素は 0 が仮定される。未使用の要素を参照した場合も 0 が返される。

----------関連する定数値----------
extern const int BDDV_SysVarTop
extern const int BDDV_MaxLen
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar
extern const bddword BDD_MaxNode
extern const int BDD_MaxVar

----------関連する外部関数----------
extern int SOPV_NewVar(void)
（このメソッドは旧版で用いていた。なくても困らないはずである）
SOP_NewVar()と同様に、新しいリテラル（正負１組）を生成し、正リテラルの変数
番号(通称 VarID)を返す。負リテラルの VarID は正リテラルよりも１小さい数値となる。
SOP_NewVar()との違いは、VarID が 2 から始まるのではなく、(BDDV_SysVarTop + 2)
から始まる点である。ただし変数の順位(通称 level)は 1 からスタートする。
出力選択変数の level は 2 ずつ上位にシフトしていく。

extern int SOPV_NewVarOfLev(int lev)
（このメソッドは旧版で用いていた。なくても困らないはずである）
SOP_NewVarOfLev()と同様に、新しいリテラル（正負１組）を生成し、正リテラルの
変数番号(通称 VarID)を返す。負リテラルの VarID は正リテラルよりも１小さい数値
となる。SOP_NewVar()との違いは、VarID が 2 から始まるのではなく、
(BDDV_SysVarTop＋2)から始まる点である。ただし指定できる変数の順位(通称 level)は、
2 以上かつ「これまでユーザが宣言した変数の個数 +2」までである。出力選択変数の
level は 2 ずつ上位にシフトしていく。

extern SOPV operator&(const SOPV& fv, const SOPV& gv)
fv と gv の各配列要素同士の共通項(intersection)を表す SOPV オブジェクトを生成し、
それを返す。

extern SOPV operator+(const SOPV& fv, const SOPV& gv)
fv と gv の各配列要素同士の結び(union)を表す SOPV オブジェクトを生成し、
それを返す。

extern SOPV operator-(const SOPV& fv, const SOPV& gv)
各配列要素同士について、fv から gv を引いた差分集合を表す SOPV オブジェクトを
生成し、それを返す。

extern int operator==(const SOPV& fv, const SOPV& gv)
fv と gv の各配列要素同士が全て同じかどうかの真偽(1/0)を返す。

extern int operator!=(const SOPV& fv, const SOPV& gv)
fv と gv の各配列要素のうち少なくとも１つが異なるかどうかの真偽(1/0)を返す。

extern SOPV SOPV_ISOP(BDDV fv)
fv で与えられた論理関数ベクトルの各配列要素について非冗長積和形を表す
SOPV オブジェクト生成し、それを返す。記憶あふれの場合や引数に null が含まれて
いるときには null を返す。

extern SOPV SOPV_ISOP(BDDV on, BDDV dc)
オンセット配列 on、ドントケアセット配列 dc の組によって表される論理関数
ベクトルの非冗長積和形を表す SOPV オブジェクトを生成し、それを返す。記憶あふれ
の場合や引数に null が含まれているときには null を返す。

extern SOPV SOPV_ISOP2(BDDV f)
fv で与えられた論理関数ベクトルの各配列要素について非冗長積和形を表す
SOPV オブジェクト生成し、それを返す。総リテラル数がなるべく小さく
なるように、出力にインバータが自動的に挿入される。生成結果の SOPV オブ
ジェクトは、出力数を n のとき、0 ～(n-1)番目の要素が積和形を表し、n
～(2n-1)番目の要素が 1（空集合）または 0（単位元集合）を表している。
記憶あふれの場合や引数に null が含まれているときには null を返す。

extern SOPV SOPV_ISOP2(BDDV on, BDDV dc)
オンセット配列 on、ドントケアセット配列 dc の組によって表される論理関数
ベクトルの非冗長積和形を表す SOPV オブジェクトを生成し、それを返す。
総リテラル数がなるべく小さくなるように、出力にインバータが自動的に
挿入される。生成結果の SOPV オブジェクトは、出力数を n のとき、
0 ～(n-1)番目の要素が積和形を表し、n ～(2n-1)番目の要素が 1（空集合）
または 0（単位元集合）を表している。記憶あふれの場合や引数に null が
含まれているときには null を返す。

（再掲）
extern void BDDV_Init(bddword init, bddword limit)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)

----------公開クラスメソッド----------
SOPV::SOPV(void)
基本 constructer。初期値として空集合（恒偽式）を表す SOPV オブジェクトを
生成する。

SOPV::SOPV(const SOPV& fv)
引数 fv を複製する constructer。

SOPV::SOPV(const ZBDDV& fv)
内部表現の ZBDDV 表現 fv を引数として、それを複製する constructer。

SOPV::SOPV(const SOP& f, int location = 0)
第 location 番目の配列要素が f で、それ以外の要素が空集合となっている
SOPV オブジェクトを生成する constructer。f が null のときは、location の値に
関わらず、長さ 1 の null となる。

SOPV::~SOPV(void)
destructer。

SOPV& SOPV::operator=(const SOPV& fv)
自分自身に fv を代入し、関数値として fv を返す。

SOPV SOPV::operator&=(const SOPV& fv)
自分自身と fv との共通項(intersection)を各配列要素毎に求め、自分自身に
代入する。記憶あふれの場合は長さ 1 の null を代入する。自分自身や引数が
null のときには null となる。

SOPV SOPV::operator+=(const SOPV& fv)
自分自身と fv の少なくとも一方に含まれる積項集合(union)を各配列要素毎に
求め、自分自身に代入する。記憶あふれの場合は長さ 1 の null を代入する。

自分自身や引数が null のときには null となる。

SOPV SOPV::operator-=(const SOPV& fv)
自分自身から fv との共通項を除いた差分集合を各配列要素毎に求め、自分自身
に代入する。記憶あふれの場合は長さ 1 の null を代入する。自分自身や引数が
null のときには null となる。

SOPV SOPV::operator<<=(int s)
自分自身の各配列要素に対して、関係する全てのリテラルを、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した積項集合の
配列を生成し、自分自身に代入する。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOPV SOPV::operator>>=(int s)
自分自身の各配列要素に対して、関係する全てのリテラルを、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した積項集合
の配列を生成し、自分自身に代入する。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOPV SOPV::operator<<(int s)
自分自身の各配列要素に対して、関係する全てのリテラルを、展開順位(level)が s ずつ
大きい（上位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した SOPV
オブジェクトを生成し、それを返す。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOPV SOPV::operator>>(int s)
自分自身の各配列要素に対して、関係する全てのリテラルを、展開順位(level)が s ずつ
小さい（下位にある）変数の変数番号(VarID)にそれぞれ書き換えて複製した SOPV
オブジェクトを生成し、それを返す。また演算結果を関数値として返す。s は偶数で
なければならない。実行結果において未定義のリテラルが必要になるような s を
与えてはならない。必要なリテラルはあらかじめ宣言しておくこと。記憶あふれの
場合は、null を表すオブジェクトを返す。自分自身が null のときは何もしない。
s に負の値を指定することはできない。

SOPV SOPV::And0(int var)
自分自身の各配列要素の各積項に、変数番号 var の負リテラルを追加した積項集合の
配列を表す SOPV オブジェクトを生成し、それを返す。ただし、同じ番号の正リテラルを
すでに含む項は消される。記憶あふれの場合や自分自身が null のときには null を返す。

SOPV SOPV::And1(int var)
自分自身の各配列要素の各積項に、変数番号 var の正リテラルを追加した積項集合の
配列を表す SOPV オブジェクトを生成し、それを返す。ただし、同じ番号の負リテラルを
すでに含む項は消される。記憶あふれの場合や自分自身が null のときには null を返す。

SOPV SOPV::Factor0(int var)
自分自身の各配列要素について、変数番号 var の負リテラルで割ったときの商に
相当する積項集合の配列を返す。すなわち、該当リテラルを含む項をすべて取りだし、
その各項から該当リテラルを取り除いた SOPV オブジェクトを生成し、それを返す。
記憶あふれの場合や自分自身が null のときには null を返す。

SOPV SOPV::Factor1(int var)
自分自身の各配列要素について、変数番号 var の正リテラルで割ったときの商に
相当する積項集合の配列を返す。すなわち、該当リテラルを含む項をすべて取りだし、
その各項から該当リテラルを取り除いた SOPV オブジェクトを生成し、それを返す。
記憶あふれの場合や自分自身が null のときには null を返す。

SOPV SOPV::FactorD(int var)
自分自身の各配列要素の積項のうち、変数番号 var の正・負リテラルをどちらも
含まない項からなる SOPV オブジェクトを生成し、それを返す。記憶あふれの場合や
自分自身が null のときには null を返す。

int SOPV::Top(void)
自分自身の各配列要素が含んでいるリテラルの中で、最上位の展開順位を持つ
リテラル（正リテラル）の変数番号を返す。必ず偶数になる。null に対しては 0 を返す。

bddword SOPV::Size(void)
自分自身のグラフのノード数を返す。null に対しては 0 を返す。出力選択変数に
関する節点は含まない。

bddword SOPV::Cube(void)
自分自身の積項数（AND ゲート数）を返す。

bddword SOPV::Lit(void)
自分自身の総リテラル数（AND ゲートの総ファンイン数）を返す。

void SOPV::Print(void)
インデックスの値、最上位のリテラル番号、ノード数、積項数、リテラル数の
情報を標準出力に出力する。

int SOPV::PrintPla(void)
自分自身が表す集合を表形式（pla format）で標準出力に出力する。関数の値
は、通常 0 を返す。null を与えた場合や記憶あふれの場合は出力を中断し、1 を
返す。

SOPV SOPV::Mask(int start, int length = 1)
自分自身の第 start 番目から 第(start+length-1)番目までの配列要素を
残し、その他の配列要素を 0（空集合）とした SOPV オブジェクトを生成し、
それを返す。

SOP SOPV::GetSOP(int ix)
自分自身の第 ix 番目の配列要素を返す。

ZBDDV SOPV::GetZBDDV(void)
内部表現の ZBDDV を複製し、それを返す。

int SOPV::Last(void)
自分自身の（意味のある）配列要素の中の、最大の要素番号を返す。

SOPV SOPV::Swap(int, int)
自分自身の各配列要素に対して、変数番号 var1 と var2 のリテラルを入れ換えた
ときの積項集合の配列を表す SOPV オブジェクトを生成し、それを返す。
var1, var2 は正リテラルの番号（偶数）を指定するだけで、負リテラルも同時に
入れ替えが実行される。引数は level ではなく、変数番号で与えることに注意。
記憶あふれの場合は、null を表すオブジェクトを返す。自分自身が null のときは、
null を返す。

```
********************************************************************************
クラス名: PiDD  ---順列集合を表現するクラス
********************************************************************************
ヘッダーファイル名: "PiDD.h"
ソースファイル名: PiDD.cc
内部から呼び出しているクラス: BDD, ZBDD

順列集合を表す PiDD を扱うクラスである。順列集合は順列の和集合であり、
順列は互換演算の積集合である。PiDD_NewVar()を用いて変数を宣言することにより、
PiDD のアイテム番号が 1 つずつ割り当てられる。

（使用例）
int v1 = PiDD_NewVar();
int v2 = PiDD_NewVar();
int v3 = PiDD_NewVar();
int v4 = PiDD_NewVar();
PiDD P1 = PiDD(1).Swap(v1, v2);
PiDD P2 = PiDD(1).Swap(v2, v3);
PiDD P3 = P1 + P2;
```

```
PiDD P4 = P1.Swap(v3, v4) + P3;
P3.Print();
P4.Print();
```

----------関連する定数値----------

```
extern const int PiDD_MaxVar 254
```

----------関連する外部関数----------

```
extern int PiDD_NewVar(void)
```

新しいアイテム変数を生成し変数番号（PiVarID）を返す。
PiVarID は 1 から始まる整数で、PiDD_NewVar()を 1 回実行するごとに 1 ずつ
大きな値が返る。その値を X とすると、実際には(X,1)(X,2)…(X,X-1)の互換を
区別するための ZBDD の ID が内部で自動的に確保される。
アイテム変数の個数が最大値 PiDD_MaxVar を超えるとエラーを出力して異常終了する。

```
extern int PiDD_VarUsed(void)
```

これまでに宣言したアイテム変数の個数を返す。

```
extern PiDD operator&(const PiDD& P, const PiDD& Q)
```

P と Q の交わり(intersection)を表す PiDD オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

```
extern PiDD operator+(const PiDD& P, const PiDD& Q)
```

P と Q の結び(union)を表す PiDD オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

```
extern PiDD operator-(const PiDD& P, const PiDD& Q)
```

P から Q を引いた差集合(difference)を表す PiDD オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

```
extern int operator==(const PiDD& P, const PiDD& Q)
```

P と Q が同じ集合かどうかの真偽(1/0)を返す。

```
extern int operator!=(const PiDD& P, const PiDD& Q)
```

P と Q が異なる集合かどうかの真偽(1/0)を返す。

```
extern PiDD operator*(const PiDD& P, const PiDD& Q)
```

P と Q の直積を表す PiDD オブジェクトを生成し、それを返す。
記憶あふれの場合や引数に null が含まれているときには null を返す。

（再掲）

```
void BDD_Init(bddword init, bddword limit)
extern int BDD_LevOfVar(int v)
extern int BDD_VarOfLev(int lev)
extern int BDD_VarUsed(void)
extern int BDD_TopLev(void)
extern bddword BDD_Used(void)
extern void BDD_GC(void)
```

----------公開クラスメソッド----------

```
PiDD::PiDD(void)
```

基本 constructer。初期値として空集合を表す PiDD オブジェクトを生成する。

```
PiDD::PiDD(int val)
```

定数式を作り出す constructer。val == 0 ならば空集合、val > 0 ならば恒等順列
だけからなる順列集合、
val < 0 ならば null を表す PiDD オブジェクトを生成する。

```
PiDD::PiDD(const PiDD& P)
```

引数 P を複製する constructer。

```
PiDD::PiDD(const ZBDD& zbdd)
```

内部表現の ZBDD 表現 zbdd を引数として、それを複製する constructer。

```
PiDD::~PiDD(void)
```

destructer。

```
PiDD& PiDD::operator=(const PiDD& P)
```

自分自身に P を代入し、P を返す。

```
PiDD PiDD::operator&=(const PiDD& P)
```

自分自身と P との交わり(intersection)を求め、自分自身に代入する。

記憶あふれの場合は null を代入する。自分自身や引数が null のときには null となる。

```
PiDD PiDD::operator+=(const PiDD& P)
```

自分自身と P との結び(union)を求め、自分自身に代入する。記憶あふれの場合は
null を代入する。自分自身や引数が null のときには null となる。

```
PiDD PiDD::operator-=(const PiDD& P)
```

自分自身から P を引いた差集合(difference)を求め、自分自身に代入する。記憶あふれの
場合は null を代入する。自分自身や引数が null のときには null となる。

```
PiDD PiDD::operator*=(const PiDD& P)
```

自分自身と P の直積を求め、自分自身に代入する。記憶あふれの場合は null を代入する。
自分自身や引数が null のときには null となる。

```
PiDD PiDD::Swap(int x, int y)
```

自分自身が含む各順列に対して番号 x と y のアイテムを互換した順列からなる
順列集合の PiDD オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が
null のときには null を返す。x または y が、0 以下であるか宣言されていない大きな番号
の場合は、エラーを出力して異常終了する。

```
PiDD PiDD::Cofact(int x, int y)
```

自分自身が含む各順列の中で、番号 x のアイテムが番号 y のアイテムに移動するような順
列のみを取り出して、さらに x と y を互換して得られる順列からなる順列集合を表す PiDD
オブジェクトを生成し、それを返す。記憶あふれの場合や自分自身が null のときには null
を返す。x == y の場合は、x が動かない順列のみが抽出される。x または y が、0 以下で
あるか宣言されていない大きな番号の場合は、エラーを出力して異常終了する。

```
int PiDD::TopX(void)
```

自分自身のグラフが含んでいる順列集合の中で、最上位の展開順位を持つ互換を(X,Y)とす
ると、番号 X を返す。null に対しては 0 を返す。

```
int PiDD::TopY(void)
```

自分自身のグラフが含んでいる順列集合の中で、最上位の展開順位を持つ互換を(X,Y)とす
ると、番号 Y を返す。null に対しては 0 を返す。

```
int PiDD::TopLev(void)
```

自分自身のグラフが含んでいる順列集合の中で、最上位の展開順位を持つ互換に割り当て
られている BDD の level 番号を返す。null に対しては 0 を返す。

```
bddword PiDD::Size(void)
```

自分自身のグラフの節点数を返す。null に対しては 0 を返す。

```
bddword PiDD::Card(void)
```

自分自身が含む順列の個数を返す。null に対しては 0 を返す。

```
void PiDD::Print(void)
```

インデックスの値、最上位の変数番号、ノード数、順列の個数、互換の総数の
情報を標準出力に出力する。

```
void PiDD::Enum(void)
```

自分自身が表す順列集合を、順列を列挙する形式で標準出力に出力する。

```
void PiDD::Enum2(void)
```

自分自身が表す順列集合を、互換標準形を列挙する形式で標準出力に出力する。

```
ZBDD PiDD::GetZBDD(void)
```

内部表現の ZBDD を複製したオブジェクトを生成して、それを返す。