

\*\*\*\*\* C++版 BDD パッケージ (SAPPORO-1.91) ドキュメント \*\*\*\*\*

著者: 湊 真一 京都大学 大学院情報学研究科

最新更新日: 2021.9.2

\*\*\*\*\*

## 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  
|     |--- BtoI           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
-- CtoI	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
-- SeqBDD	系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-----

```
int 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ない、limit を越えてはならない。limit は、256 以上でなければならない、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。

```
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。

```
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。

```
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。

```
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。

```
int BDD_VarUsed(void)
```

現在までに宣言した入力変数の個数を返す。

```
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。

```
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。

```
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。

`bddword BDD_CacheInt(unsigned char op, bddword f, bddword g)`

f と g の演算結果が非負整数値のとき、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、null に相当する数値 (BDD\_MaxNode よりも約 2 倍大きな数値で BDD(-1).GetID() で得られる値) 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。

`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。

`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。

`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。

`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。

`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。

`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。

```
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。

```
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。

```
int BDD_Imply(const BDD& f, const BDD& g)
```

f と g の包含性判定を行う。すなわち、 $(\sim f \mid g)$  が恒真かどうかを調べる。

$(\sim f \mid g)$  の BDD オブジェクトを生成せずに判定だけを行うので、 $(\sim f \mid g)$  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。

```
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。

```
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。

```
void BDDerr(char *msg)
```

```
void BDDerr(char *msg, bddword key)
```

```
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 constructor。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 constructor。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 constructor。

BDD::~~BDD(void)

destructor。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切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(Var ID)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(Var ID)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) const

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) const

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(Var ID)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) const

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(Var ID)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) const

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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-----

int 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。

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。

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。

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。

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。

`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。

`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。

`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）。

`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）。

`int BDDV_Imply(BDDV fv, BDDV gv)`

fv と gv の包含性判定を行う。すなわち、 $(\sim fv \mid gv)$  が全ての要素について

恒真かどうかを調べる。 $(\sim fv \mid gv)$  の BDDV オブジェクトを生成せずに

判定だけを行うので、 $(\sim fv \mid gv)$  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。

`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。

`BDDV BDDV_ImportPla(FILE *strm = stdin, int sopf = 0)`

strm で指定するファイルから ESPRESSO フォーマットの PLA データを読み込み、BDDV オブジェクトを生成して、それを返す。ESPRESSO フォーマットでは、`.i .o .type .e` のキーワードのみサポートする。生成される BDDV オブジェクトの要素数は、入力された PLA データの出力数のちょうど 2 倍になっており、前半部分が onset 関数、後半部分が dcset 関数を表現している。sopf フラグが 0 以外の場合は、SOP クラスとデータ変換がしやすいように、偶数番号の VarID を使用する。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。

`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。

`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）。

`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)

`int BDD_NewVar(void)`

`int BDD_NewVarOfLev(int lev)`

`int BDD_LevOfVar(int v)`

`int BDD_VarOfLev(int lev)`

`int BDD_VarUsed(void)`

`int BDD_TopLev(void)`

`bddword BDD_Used(void)`

`void BDD_GC(void)`

-----公開クラスメソッド-----

`BDDV::BDDV(void)`

基本 constructor。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 constructor。

`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 constructor。`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)`

destructor。

`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(`Var ID`)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 (Var ID) 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) const`

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) const`

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 (Var ID) 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) const`

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 (Var ID) 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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

配列長を返す。

`void BDDV::Print(void) const`

内部情報を標準出力に出力する。

\*\*\*\*\*

クラス名: BtoI --- 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: "BtoI.h"

ソースファイル名: BtoI.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、関数を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、BtoI (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-----

```
int operator==(const BtoI& fv, const BtoI& gv)
2つの BtoI 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。BtoI_EQ() とは意味が違うので注意。
```

```
int operator!=(const BtoI& fv, const BtoI& gv)
2つの BtoI 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。BtoI_NE() とは意味が違うので注意。
```

`Btoi operator+(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvの算術和を表すBtoi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。

`Btoi operator-(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvの算術差を表すBtoi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。

`Btoi operator*(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvの算術積を表すBtoi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。

`Btoi operator/(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvの整数除算の商を表すBtoi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)  $10 / (-3) = -3$

`Btoi operator%(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvの算術和を表すBtoi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)  $-10 \% 3 = -1$

`Btoi operator&(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvのビット論理積を表すBtoi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。

`Btoi operator|(const Btoi& fv, const Btoi& gv)`

2つのBtoiオブジェクトfv, gvのビット論理和を表すBtoi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。

`BtoI operator^(const BtoI& fv, const BtoI& gv)`

2つの BtoI オブジェクト fv, gv のビット排他論理和を表す BtoI 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。

`BtoI BtoI_ITE(BDD f, BtoI gv, BtoI 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 BtoI 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。

`BtoI BtoI_ITE(BtoI fv, BtoI gv, BtoI hv)`

fv が 0 以外の場合は 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 BtoI 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。

`BtoI BtoI_EQ(BtoI fv, BtoI 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 BtoI 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。

`BtoI BtoI_NE(BtoI fv, BtoI 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 BtoI 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。

`BtoI BtoI_GT(BtoI fv, BtoI 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 BtoI 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。

`BtoI BtoI_LT(BtoI fv, BtoI 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 BtoI 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。

BtoI BtoI\_GE(BtoI fv, BtoI gv)

fv が gv より大きいと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 BtoI 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。

BtoI BtoI\_LE(BtoI fv, BtoI gv)

fv が gv より小さいと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 BtoI 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。

BtoI BtoI\_atoi(char\* s)

s の指す数字文字列を数値に変換し、定数関数の BtoI 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)

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

-----公開クラスメソッド-----

```
BtoI::BtoI(void)
```

基本 constructor。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。

`BtoI::BtoI(const BtoI& fv)`

BtoI オブジェクトのコピーを生成する constructor。

`BtoI::BtoI(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 BtoI オブジェクトを生成する constructor。 $f$  が null の場合は1ビットの null からなるオブジェクト `BtoI(BDD(-1))` を生成。

`BtoI::BtoI(int n)`

定数関数（常に整数値  $n$  を返す）を表す BtoI オブジェクトを生成する constructor。

`BtoI::BtoI(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。

`BtoI::~~BtoI(void)`

destructor。

`BtoI& BtoI::operator=(const BtoI& fv)`

$fv$  を自分自身に代入し、それを値として返す。

`BtoI BtoI::operator+=(const BtoI& 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。

`BtoI BtoI::operator-=(const BtoI& 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。

`BtoI BtoI::operator*=(const BtoI& 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。

`BtoI BtoI::operator/=(const BtoI& 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`

`BtoI BtoI::operator%=(const BtoI& 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`

`BtoI BtoI::operator&=(const BtoI& 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。

`BtoI BtoI::operator|=(const BtoI& 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。

`BtoI BtoI::operator^=(const BtoI& 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。

`BtoI BtoI::operator<<=(const BtoI& 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。

`BtoI BtoI::operator>>=(const BtoI& 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。

`BtoI BtoI::operator-(void) const`



自分自身の補数を表す BtoI 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。

BtoI BtoI::operator~(void) const

自分自身のビット反転（1の補数）を表す BtoI 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。

BtoI BtoI::operator!(void) const

自分自身の論理否定を表す BtoI 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。  
BtoI\_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。

BtoI BtoI::operator<<(const BtoI& fv) const

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 BtoI  
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。

BtoI BtoI::operator>>(const BtoI& fv) const

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 BtoI  
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。

BtoI BtoI::UpperBound(void) const

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 BtoI 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。

BtoI BtoI::UpperBound(BDD f) const

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 BtoI 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。

`BtoI BtoI::LowerBound(void) const`

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 BtoI 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。

`BtoI BtoI::LowerBound(BDD) const`

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 BtoI 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。

`BtoI BtoI::At0(int var) const`

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 BtoI 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。

`BtoI BtoI::At1(int) const`

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 BtoI 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。

`BtoI BtoI::Cofact(BtoI fv) const`

自分自身の表す関数において、fv = 0 のときを don't care とみなして単純化した関数を表す BtoI 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 BtoI::Top(void) const`

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 BtoI::GetSignBDD(void) const`

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 BtoI::GetBDD(int ix) const
```

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 BtoI::GetMetaBDDV(void) const
```

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 BtoI::Len(void) const
```

自分自身の桁数を返す。

```
int BtoI::GetInt(void) const
```

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 BtoI::StrNum10(char* s) const
```

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 BtoI::StrNum16(char* s) const
```

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 BtoI::Size() const
```

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 BtoI::Print(void) const
```

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)

```
int v1 = BDD_NewVar();
```

```
int v2 = BDD_NewVar();
```

```
int v3 = BDD_NewVar();
```

```
int v4 = BDD_NewVar();
```

```
ZBDD x = ZBDD(1).Change(v1);
```

```
ZBDD y = ZBDD(1).Change(v2);
```

```
ZBDD f = x + y;
```

```
ZBDD g = f.Change(v3) + f.Change(v4);
```

```
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-----

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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)
```

```
int BDD_NewVar(void)
```

```
int BDD_NewVarOfLev(int lev)
```

```
int BDD_LevOfVar(int v)
```

```
int BDD_VarOfLev(int lev)
```

```
int BDD_VarUsed(void)
```

```
int BDD_TopLev(void)
bddword BDD_Used(void)
void BDD_GC(void)
```

-----公開クラスメソッド-----

```
ZBDD::ZBDD(void)
```

constructor。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 constructor。  $v == 0$  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 constructor。

```
ZBDD::~~ZBDD(void)
```

destructor。

```
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 (Var ID) 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 (Var ID) 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) const

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 (Var ID) 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) const

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 (Var ID) 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) const

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) const

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) const

`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) const

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) const

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) const

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) const

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) const

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) const

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。

ZBDD ZBDD::Always(void) const

自分自身の集合に属する組合せ全てに共通して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::IsPoly(void) const

自分自身の集合に組合せが複数個含まれるかどうかの真偽を返す。

`int ZBDD::Top(void) const`

自分自身の組合せ集合に関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) const`

集合を一意に表現する 1-word の識別番号を返す。

`bddword ZBDD::Size(void) const`

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) const`

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。

`char* ZBDD::CardMP16(char* s) const`

自分自身が表す集合の要素数 (cardinality) を最大 16 ワード長までの多倍長整数でカウントする。結果は 16 進数文字列として s から始まる記憶領域に格納する。(詳細は C 言語版の `bddcardmp16` を参照)

`bddword ZBDD::Lit(void) const`

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。

`bddword ZBDD::Len(void) const`

自分自身が表す集合の中で、最もアイテム数を多く含む組合せを探し出して、そのアイテム数を返す。null に対しては 0 を返す。

`void ZBDD::Export(FILE *strm = stdout) const`

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。

`void ZBDD::PrintPla(void) const`

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) const`

自分自身のグラフを、X-Window に描画する。

`void ZBDD::XPrint0(void) const`

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) const
```

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。

```
bddword ZBDD::ZLev(int lev, int last = 0) const
```

自分自身の ZBDD について、最上位節点から 0-枝を順にたどって行って、アイテム変数のレベルがちょうど lev となっている節点があれば、それを最上位節点とする ZBDD オブジェクトをコピーして返す。変数レベルがちょうど lev となっている節点が無ければ、last がゼロのときは、lev 以下となる最初の節点を最上位節点とする ZBDD オブジェクトをコピーして返す。last が非ゼロのときは、lev 以上である最後の節点を最上位節点とする ZBDD オブジェクトをコピーして返す。null に対しては null を返す。なお、ZBDD オブジェクトに対してあらかじめ SetZSkip() を 1 回実行しておく、補助リンクのおかげで ZLev メソッドが高速に行える。

```
void ZBDD::SetZSkip(void) const
```

自分自身の ZBDD について、0-枝を高速にたどるための補助リンクを張る。それ以外には基本的に変化しない。

```
bddword ZBDD::Intersec(ZBDD g) const
```

自分自身と g との共通集合を表す ZBDD オブジェクトを生成し、それを返す。どちらかが null ならば null を返す。あらかじめ SetZSkip() を 1 回実行しておく、補助リンクのおかげで高速に実行できる。特に、自分自身の ZBDD オブジェクトが含むアイテム変数の個数が非常に多く、g に出現するアイテム変数の個数が非常に少ないときに有効である。

\*\*\*\*\*

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-----

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。

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。

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。

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。

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。

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)

```
void BDDV_Init(bddword init, bddword limit)
```

```
int BDD_NewVar(void)
```

```
int BDD_NewVarOfLev(int lev)
```

```
int BDD_LevOfVar(int v)
```

```
int BDD_VarOfLev(int lev)
```

```
int BDD_VarUsed(void)
```

```
int BDD_TopLev(void)
```

```
bddword BDD_Used(void)
```

```
void BDD_GC(void)
```

-----公開クラスメソッド-----

```
ZBDDV::ZBDDV(void)
```

基本 constructor。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 constructor。

```
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 constructor。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)
```

destructor。

```
ZBDDV& ZBDDV::operator=(const ZBDDV& fv)
```

自分自身に fv を代入し、fv を返す。

```
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 とする。

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) const

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) const

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) const

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) const

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) const

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) const

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) const

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

自分自身のグラフを、X-Window に描画する。

```
void ZBDDV::XPrint0(void) const
```

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: CtoI --- 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: "CtoI.h"

ソースファイル名: CtoI.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、CtoI\_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-----

```
int operator==(const CtoI& a, const CtoI& b)
```

2つの CtoI 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。CtoI\_EQ() とは意味が違うので注意。

```
int operator!=(const CtoI& a, const CtoI& b)
```

2つの CtoI 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。CtoI\_NE() とは意味が違うので注意。

```
CtoI operator+(const CtoI& a, const CtoI& b)
```

2つの CtoI 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 CtoI 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。

CtoI operator-(const CtoI& a, const CtoI& b)

2つの CtoI 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 CtoI 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。

CtoI operator\*(const CtoI& a, const CtoI& b)

2つの CtoI オブジェクト a, b の算術積を表す CtoI 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 ( $v \times v = v$ ) 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、 $(x + 2y)(x + 3y) = x^2 + 6xy + 5y^2$  となる。

CtoI operator/(const CtoI& a, const CtoI& b)

2つの CtoI オブジェクト a, b の整数除算の商を表す CtoI 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、 $(20x + 12xy + 10y + 8) / (3x + 2)$  の場合、  
 $(20x + 12xy + 10y + 8)/(3x) = 6 + 4y$ ,  
 $(20x + 12xy + 10y + 8)/2 = 10x + 6xy + 5y + 4$   
なので、全体の商は  $4y + 4$  となる。

CtoI operator%(const CtoI& a, const CtoI& b)

2つの CtoI オブジェクト a, b の整数除算の剰余を表す CtoI 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)  $-10 \% 3 = -1$ 。

CtoI CtoI\_Null(void)

エラーや記憶あふれを表す CtoI 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。

CtoI CtoI\_ITE(CtoI a, CtoI b, CtoI 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 CtoI 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。

CtoI CtoI\_EQ(CtoI a, CtoI 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 CtoI 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。

CtoI CtoI\_NE(CtoI a, CtoI 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 CtoI 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。

CtoI CtoI\_GT(CtoI a, CtoI 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 CtoI 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。

CtoI CtoI\_GE(CtoI a, CtoI 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 Ctoi 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。

Ctoi Ctoi\_LT(Ctoi a, Ctoi 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 Ctoi 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。

Ctoi Ctoi\_LE(Ctoi a, Ctoi 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 Ctoi 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。

Ctoi Ctoi\_Max(Ctoi a, Ctoi 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 Ctoi 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。

Ctoi Ctoi\_Min(Ctoi a, Ctoi 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 Ctoi 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。

Ctoi Ctoi\_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 Ctoi 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。

Ctoi Ctoi\_Intsec(Ctoi a, Ctoi b)

2つの Ctoi 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 Ctoi 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。

Ctoi Ctoi\_Union(Ctoi a, Ctoi b)

2つの Ctoi 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 Ctoi 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。

Ctoi Ctoi\_Diff(Ctoi a, Ctoi b)

2つの Ctoi 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 Ctoi 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。

Ctoi\_Meet(Ctoi a, Ctoi 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 Ctoi 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 次に変換して残したものと考えてよい。例えば、 $\text{Meet}(x + 2xy + 3y, x + y + 1)$

$= x + 1 + 1 + 2x + 2y + 2 + 3 + 3y + 3 = 3x + 5y + 10$  となる。

Ctoi Ctoi\_LcmA(char \*fname1, char \*fname2, int th)

fname1 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 Ctoi オブジェクトを生成し、それを返す。fname2 に NULL 以外が指定された場合は、変数順序ファイルを読み込んで、その順序で LCM を実行する。記憶あふれの場合は 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。(LCM 関連のメソッドは別途インストールする必要あり)

Ctoi Ctoi\_LcmC(char \*fname1, char \*fname2, int th)

fname1 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 `CtoI` オブジェクトを生成し、それを返す。`fname2` に `NULL` 以外が指定された場合は、変数順序ファイルを読み込んで、その順序で LCM を実行する。記憶あふれの場合は `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。(LCM 関連のメソッドは別途インストールする必要あり)

`CtoI CtoI_LcmM(char *fname1, char *fname2, int th)`

`fname1` 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 `CtoI` オブジェクトを生成し、それを返す。`fname2` に `NULL` 以外が指定された場合は、変数順序ファイルを読み込んで、その順序で LCM を実行する。記憶あふれの場合は `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。(LCM 関連のメソッドは別途インストールする必要あり)

`CtoI CtoI_LcmAV(char *fname1, char *fname2, int th)`

`CtoI_LcmA` とほとんど同じだが、それぞれのアイテム組合せの出現回数を保持した `CtoI` オブジェクトを生成し、それを返す。(LCM 関連のメソッドは別途インストールする必要あり)

`CtoI CtoI_LcmCV(char *fname1, char *fname2, int th)`

`CtoI_LcmC` とほとんど同じだが、それぞれのアイテム組合せの出現回数を保持した `CtoI` オブジェクトを生成し、それを返す。(LCM 関連のメソッドは別途インストールする必要あり)

`CtoI CtoI_LcmMV(char *fname1, char *fname2, int th)`

`CtoI_LcmM` とほとんど同じだが、それぞれのアイテム組合せの出現回数を保持した `CtoI` オブジェクトを生成し、それを返す。(LCM 関連のメソッドは別途インストールする必要あり)

(再掲)

`void BDDV_Init(bddword init, bddword limit)`

`int BDD_NewVar(void)`

`int BDD_NewVarOfLev(int lev)`

`int BDD_LevOfVar(int v)`

`int BDD_VarOfLev(int lev)`

`int BDD_VarUsed(void)`

`int BDD_TopLev(void)`

`bddword BDD_Used(void)`

void BDD\_GC(void)

-----公開クラスメソッド-----

CtoI::CtoI(void)

基本 constructor。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。

CtoI::CtoI(const CtoI& a)

CtoI オブジェクトのコピーを生成する constructor。

CtoI::CtoI(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。

CtoI::CtoI(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 CtoI オブジェクトを生成する constructor。

CtoI::~~CtoI(void)

destructor。

CtoI& CtoI::operator=(const CtoI& a)

a を自分自身に代入し、それを値として返す。

int CtoI::Top(void) const

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。CtoI 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 CtoI::TopItem(void) const

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 CtoI::TopDigit(void) const`

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) const`

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) const`

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。

`CtoI CtoI::AffixVar(int var) const`

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 CtoI 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。CtoI 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。

`CtoI CtoI::Factor0(int var) const`

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 CtoI 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。CtoI 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。

`CtoI CtoI::Factor1(int var) const`

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 CtoI 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。CtoI 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。

`CtoI CtoI::FilterThen(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::FilterElse(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::FilterRestrict(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::FilterPermit(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::FilterPermitSym(int n) const`

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 CtoI 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。

`CtoI CtoI::Support(void) const`

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 CtoI 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。

`CtoI CtoI::NonZero(void) const`

自分自身に含まれる組合せ要素をすべて抽出した CtoI 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。

`CtoI CtoI::Digit(int d) const`

自分自身の第 d 桁目の組合せ集合を表す CtoI 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。

`CtoI CtoI::EQ_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::NE_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::GT_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::GE_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::LT_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::LE_Const(CtoI a) const`

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 CtoI 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。

`CtoI CtoI::MaxVal(void) const`

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 CtoI 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。

`CtoI CtoI::MinVal(void) const`

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 CtoI 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。

`CtoI CtoI::TotalVal(void) const`

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 CtoI 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 CtoI::GetZBDD(void) const`

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。

`CtoI CtoI::CountTerms(void) const`

自分自身が含む組合せ要素の総数を表す、定数項の CtoI 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。

CtoI CtoI::Abs(void) const

自分自身の組合せ集合の整数値を絶対値に書き換えた CtoI 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。

CtoI CtoI::Sign(void) const

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 CtoI 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。

CtoI CtoI::operator -(void) const

自分自身の組合せ集合の整数値の正負を反転させた CtoI 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。

CtoI CtoI::TimesSysVar(int var) const

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 CtoI 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。CtoI 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。

CtoI CtoI::DivBySysVar(int var) const

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 CtoI 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。CtoI 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。

CtoI CtoI::ShiftDigit(int power) const

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 CtoI 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。

`CtoI CtoI::operator+=(const CtoI& 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。

`CtoI CtoI::operator-=(const CtoI& 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。

`CtoI CtoI::operator*=(const CtoI& 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。

`CtoI CtoI::operator/=(const CtoI& 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。

`CtoI CtoI::operator%=(const CtoI& 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 CtoI::Size() const`

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 CtoI::GetInt(void) const`

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 CtoI::StrNum10(char* s) const`

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 CtoI::StrNum16(char* s) const
```

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 CtoI::PutForm(void) const
```

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 CtoI::Print(void) const
```

自分自身の内部データ情報を標準出力に出力する。

```
void CtoI::XPrint(void) const
```

自分自身の内部データ情報を X-Window に描画する。

```
void CtoI::XPrint0(void) const
```

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に表示する。）

\*\*\*\*\*

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(2k)番、負リテラル(2k - 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-----

```
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。

```
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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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)

int BDD\_LevOfVar(int v)

int BDD\_VarOfLev(int lev)

int BDD\_VarUsed(void)

int BDD\_TopLev(void)

bddword BDD\_Used(void)

void BDD\_GC(void)

-----公開クラスメソッド-----

SOP::SOP(void)

基本 constructor。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 constructor。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 constructor。

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 constructor。

SOP::~~SOP(void)

destructor。

`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 (Var ID) 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 (Var ID) 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) const

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 (Var ID) 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) const

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 (Var ID) 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) const

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) const

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) const

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) const

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) const

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) const

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) const

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) const

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) const

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) const
```

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) const
```

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) const
```

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) const
```

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) const
```

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) const
```

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) const
```

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) const
```

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) const

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) const

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) const

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-----

```
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)

SOPV\_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。SOPV\_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。

```
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)

SOPV\_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。SOPV\_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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。総リテラル数になるべく小さく

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 0 (空集合) または 1 (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。

```
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。

総リテラル数に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 0 (空集合)

または 1 (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)

```
void BDDV_Init(bddword init, bddword limit)
```

```
int SOP_NewVar(void)
```

```
int SOP_NewVarOfLev(int lev)
```

```
int BDD_LevOfVar(int v)
int BDD_VarOfLev(int lev)
int BDD_VarUsed(void)
int BDD_TopLev(void)
bddword BDD_Used(void)
void BDD_GC(void)
```

-----公開クラスメソッド-----

```
SOPV::SOPV(void)
```

基本 constructor。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 constructor。

```
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 constructor。

```
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 constructor。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)
```

destructor。

```
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(Var ID)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(Var ID)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) const

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(Var ID)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) const

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 (Var ID) 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) const

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) const

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) const

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) const

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) const

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) const

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) const`

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) const`

自分自身の積項数（AND ゲート数）を返す。

`bddword SOPV::Lit(void) const`

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) const`

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) const`

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) const`

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) const`

自分自身の第 ix 番目の配列要素を返す。

`ZBDDV SOPV::GetZBDDV(void) const`

内部表現の ZBDDV を複製し、それを返す。

`int SOPV::Last(void) const`

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) const

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 = BDD_NewVar();
int v2 = BDD_NewVar();
int v3 = BDD_NewVar();
int v4 = BDD_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-----

```
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  $(X, 1) (X, 2) \cdots (X, X-1)$  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。

```
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。

```
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。

```
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。

```
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。

```
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。

```
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。

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)

int BDD\_NewVar(void)

int BDD\_NewVarOfLev(int lev)

int BDD\_LevOfVar(int v)

int BDD\_VarOfLev(int lev)

int BDD\_VarUsed(void)

int BDD\_TopLev(void)

bddword BDD\_Used(void)

void BDD\_GC(void)

-----公開クラスメソッド-----

PiDD::PiDD(void)

基本 constructor。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 constructor。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 constructor。

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 constructor。

PiDD::~~PiDD(void)

destructor。

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) const

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) const

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。

PiDD PiDD::Odd(void) const

自分自身が含む順列の中から奇置換のみを取り出した順列集合を表す 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。

PiDD PiDD::Even(void) const

自分自身が含む順列の中から偶置換のみを取り出した順列集合を表す 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。

`PiDD PiDD::SwapBound(int n) const`

自分自身が含む順列の中から n 回以下の互換で生成可能なものだけを取り出した順列集合を表す 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。

`int PiDD::TopX(void) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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) const`

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。

\*\*\*\*\*

クラス名: SeqBDD ---系列集合を表現するクラス

\*\*\*\*\*

ヘッダーファイル名: "SeqBDD.h"

ソースファイル名: SeqBDD.cc

内部から呼び出しているクラス: BDD, ZBDD

系列集合を表す SeqBDD を扱うクラスである。

(使用例)

```
int v1 = BDD_NewVar();
int v2 = BDD_NewVar();
int v3 = BDD_NewVar();
int v4 = BDD_NewVar();
SeqBDD S1 = SeqBDD(1).Push(v1);
SeqBDD S2 = SeqBDD(1).Push(v2).Push(v3);
SeqBDD S3 = S1 + S2;
SeqBDD S4 = S1.Push(v4) + S3;
S3.Print();
S4.Print();
```

-----関連する外部関数-----

SeqBDD operator&(const SeqBDD& F, const SeqBDD& G)

F と G の交わり(intersection)を表す SeqBDD 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。

`SeqBDD operator+(const SeqBDD& F, const SeqBDD& G)`

F と G の結び(union)を表す SeqBDD 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。

`SeqBDD operator-(const SeqBDD& F, const SeqBDD& G)`

F から G を引いた差集合(difference)を表す SeqBDD 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。

`SeqBDD operator*(const SeqBDD& F, const SeqBDD& G)`

F と G の直積集合(Cartesian Product)を表す SeqBDD 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。

`SeqBDD operator/(const SeqBDD& F, const SeqBDD& P)`

F を P で割った商(Quotient)を表す SeqBDD オブジェクトを生成し、それを返す。

P=0 のときはエラー。記憶あふれの場合や引数に null が含まれている場合には null を返す。

`SeqBDD operator%(const SeqBDD& F, const SeqBDD& P)`

F を P で割った剰余(Remainder)を表す SeqBDD オブジェクトを生成し、それを返す。

P=0 のときはエラー。記憶あふれの場合や引数に null が含まれている場合には null を返す。

`int operator==(const SeqBDD& F, const SeqBDD& G)`

F と G が同じ集合かどうかの真偽(1/0)を返す。

`int operator!=(const SeqBDD& F, const SeqBDD& G)`

F と G が異なる集合かどうかの真偽(1/0)を返す。

(再掲)

`void BDD_Init(bddword init, bddword limit)`

`int BDD_NewVar(void)`

`int BDD_NewVarOfLev(int lev)`

`int BDD_LevOfVar(int v)`

`int BDD_VarOfLev(int lev)`

`int BDD_VarUsed(void)`

```
int BDD_TopLev(void)
bddword BDD_Used(void)
void BDD_GC(void)
```

-----公開クラスメソッド-----

```
SeqBDD::SeqBDD(void)
```

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

```
SeqBDD::SeqBDD(int val)
```

定数式を作り出す constructor。val == 0 ならば空集合、val > 0 ならば空列だけからなる系列集合、val < 0 ならば null を表す PiDD オブジェクトを生成する。

```
SeqBDD::SeqBDD(const SeqBDD& F)
```

引数 F を複製する constructor。

```
SeqBDD::SeqBDD(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 constructor。

```
SeqBDD::~~SeqBDD(void)
```

destructor。

```
SeqBDD& SeqBDD::operator=(const SeqBDD& F)
```

自分自身に F を代入し、F を返す。

```
SeqBDD SeqBDD::operator&=(const SeqBDD& 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。

```
SeqBDD SeqBDD::operator+=(const SeqBDD& 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。

```
SeqBDD SeqBDD::operator-=(const SeqBDD& F)
```

自分自身から F 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。

`SeqBDD SeqBDD::operator*=(const SeqBDD& F)`

自分自身と F 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。

`SeqBDD SeqBDD::operator/=(const SeqBDD& P)`

自分自身を P で割ったときの商を求め、自分自身に代入する。記憶あふれの場合は null を代入する。P=0 のときはエラーとなる。自分自身や引数が null のときには null となる。

`SeqBDD SeqBDD::operator%=(const SeqBDD& P)`

自分自身を P で割ったときの剰余を求め、自分自身に代入する。記憶あふれの場合は null を代入する。P=0 のときはエラーとなる。自分自身や引数が null のときには null となる。

`SeqBDD SeqBDD::Offset(int var) const`

自分自身の系列集合の中で、変数番号 var の文字で始まらない系列だけを取り出した集合のオブジェクトを生成しそれを返す。null に対しては 0 を返す。範囲外の変数番号を与えるとエラー。

`SeqBDD SeqBDD::OnSet0(int var) const`

自分自身の系列集合の中で、変数番号 var の文字で始まる系列だけを取り出し、それらの 1 文字目を取り除いた系列の集合のオブジェクトを生成しそれを返す。null に対しては 0 を返す。範囲外の変数番号を与えるとエラー。

`SeqBDD SeqBDD::OnSet(int var) const`

自分自身の系列集合の中で、変数番号 var の文字で始まる系列だけを取り出した集合のオブジェクトを生成しそれを返す。null に対しては 0 を返す。範囲外の変数番号を与えるとエラー。

`SeqBDD SeqBDD::Push(int var) const`

自分自身の系列集合の各系列に対して、変数番号 var の文字を先頭に追加して得られる集合のオブジェクトを生成しそれを返す。null に対しては 0 を返す。範囲外の変数番号を与えるとエラー。

`int SeqBDD::Top(void) const`

自分自身の系列集合に含まれる系列の先頭文字の中で、最上位の変数番号を返す。null に対しては 0 を返す。

`bddword SeqBDD::Size(void) const`

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 SeqBDD::Card(void) const`

自分自身が含む系列の個数を返す。null に対しては 0 を返す。

`bddword SeqBDD::Lit(void) const`

自分自身の系列集合の総文字数を返す。null に対しては 0 を返す。

`bddword SeqBDD::Len(void) const`

自分自身のグラフの高さ（最長経路の長さ）を返す。null に対しては 0 を返す。

`void SeqBDD::PrintSeq(void) const`

自分自身が含む系列を辞書順に標準出力に出力する。

`ZBDD SeqBDD::GetZBDD(void) const`

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。

\*\*\*\*\*

クラス名: GBase ---ZBDD でパス/サイクル列挙を行うためのクラス

\*\*\*\*\*

ヘッダーファイル名: "GBase.h"

ソースファイル名: GBase.cc

内部から呼び出しているクラス: BDD, ZBDD

ZBDD でパス/サイクル列挙を行うためのクラスである。n 頂点、m 辺からなるグラフを表現する。頂点番号は 1 から n までの自然数、辺番号は 0 から m-1 までの自然数とする。

-----公開クラスメソッド-----

`GBase::GBase(void)`

constructor。初期値として空の GBase オブジェクトを生成する。

`GBase::~~GBase(void)`

destructor。副次的に確保した記憶領域を全て開放する。

`int GBase::Init(const int n, const int m)`

自分自身が空でない場合はデータを消去し記憶領域を解放する。その後、

n 頂点、m 辺からなるグラフを保持するための記憶領域を確保する。

各辺の両端点の頂点番号は 0 (null) が初期値で入っている。

各辺のコスト値は 1 が初期値で入る。正常終了の場合 0 を返し、

記憶あふれの場合は 1 を返す。

`int GBase::Pack(void)`

自分自身のグラフデータを走査し、どの辺からも参照されていない

孤立頂点を消去し、残った頂点に 1 から始まる番号を振り直して

自分自身のグラフデータを正規化する。正常終了の場合 0 を返し、

記憶あふれの場合は 1 を返す。

`int GBase::Import(FILE *fp)`

fp で参照された入力ファイルから、グラフ記述データを読み込み、

自分自身にロードする。正常終了の場合 0 を返し、記憶あふれの場合は

1 を返す。入力データの記述例は以下に示す通りで、最初の 2 行で

頂点数 n と辺数 m を指定し、その後の m 行に渡って、各辺の両端点の

頂点番号を並べる。各行で #c:350 のように辺のコストを記述できる。

コスト値は int 型の範囲を想定している。コスト記述を省略すると

default で 1 が代入される。

-----  
#n 9

#m 12

1 4 #c:20

1 2 #c:250

2 5 #c:15

2 3 #c:120



```
3 6 #c:220
4 7 #c:250
4 5 #c:85
5 8 #c:77
5 6 #c:5
6 9 #c:0
7 8 #c:-25
8 9 #c:100
-----
```

```
int GBase::SetGrid(const int x, const int y)
```

横  $x$ 、縦  $y$  の 2 次元格子グラフを生成し、自分自身にセットする。  
正常終了の場合 0 を返し、記憶あふれ等の異常終了時は 1 を返す。  
 $x$ ,  $y$  は 0 以上の自然数で、頂点の繰り返し数ではなく格子の  
繰り返し数を表す。各辺のコストは default の 1 が代入される。

```
void GBase::Print(void) const
```

自分自身の内容を標準出力に書き出す。Import メソッドで読み込み  
可能なフォーマットで出力する。

```
ZBDD SimPaths(const GS_v s, const GS_v t)
```

頂点番号  $s$  と  $t$  を結び同じ頂点を二度通らないパス (Simple Paths) を  
列挙した ZBDD を生成し、生成した ZBDD オブジェクトを返す。  
記憶あふれ等の異常終了時は -1 (NULL) を返す。GS\_v は頂点番号を  
格納するデータ型で、8bit char または 16bit short である。  
自分自身の頂点数が 2 未満のときや辺数が 1 未満のときは 0 を返す。  
範囲外の頂点番号を指定した場合は -1 (NULL) を返す。

```
ZBDD SimCycles(void)
```

同じ頂点を二度通らないサイクル (Simple Cycles) を  
列挙した ZBDD を生成し、生成した ZBDD オブジェクトを返す。  
記憶あふれ等の異常終了時は -1 (NULL) を返す。  
自分自身の頂点数が 3 未満のときや辺数が 3 未満のときは 0 を返す。

```
int GBase::BDDvarOfEdge(const GS_e ix) const
```

辺番号  $ix$  に対応する BDD/ZBDD の変数 ID を返す。

辺番号が若い方から順に ZBDD の上位に割り当てられている。

GS\_e は辺番号を格納するデータ型で 16bit short である。

```
GS_e GBase::EdgeOfBDDvar(const int var) const
```

BDD/ZBDD の変数 ID var に対応する辺番号 (0~m-1) を返す。

```
void GBase::FixEdge(const GS_e ix, const char fixcode)
```

辺番号 ix の辺を必ず使う (または決して使わない) ことを

SimPaths/SimCycle メソッドを実行する前に設定する。

使う使わないは fixcode で指定する。fixcode 定数 GS\_fix0 は、

その辺を決して使わないこと、fixcode 定数 GS\_fix1 は必ず使う

ことを表す。各辺の fixcode の初期値は 0 で、制約がないことを表す。

```
void GBase::SetHamilton(const int x)
```

SimPaths/SimCycle メソッドを呼び出す際に、すべての頂点に立ち寄るような

パス/サイクルのみに制限するかどうかを設定する。x が真であれば全ての頂点に

立ち寄るような辺集合 (ハミルトン路/閉路) だけを列挙する。初期値は偽。

```
void GBase::SetCond(ZBDD f)
```

SimPaths/SimCycle メソッドを呼び出す際に、列挙するパス/サイクルが

満たすべき条件をセットする。ZBDD により任意の制約条件を組合せ集合

として記述できる。

\*\*\*\*\*

クラス名: BDDCT --- BDD/ZBDD でコスト制約付き変数を扱うためのクラス

\*\*\*\*\*

ヘッダーファイル名: "BDDCT.h"

ソースファイル名: BDDCT.cc

内部から呼び出しているクラス: BDD, ZBDD

BDDCT は BDD Cost Table の略で BDD/ZBDD でコスト制約付き変数を扱うためのクラスである。内部で 1~n までの level の BDD 変数に対応する整数値のコストを保持するテーブルを備えていて、それに基づいて ZBDD のコスト最小または最大となる解（入力値組合せ）を求めるメソッドを提供する。

```
typedef bddcost
```

コストを表す変数型を定義している。今のところ符号付 32 ビット整数。

```
#define bddcost_null 0x7FFFFFFF
```

bddcost 型で無効値を表す定数値。

```
#define CT_STRLEN 15
```

ラベル文字列の最大長を表す定数値。

-----公開クラスメソッド-----

```
BDDCT::BDDCT(void)
```

constructor。初期値として空の BDDCT オブジェクトを生成する。

```
BDDCT::~~BDDCT(void)
```

destructor。副次的に確保した記憶領域を全て開放する。

```
int BDDCT::Size(void) const
```

自分自身が記録している変数レベルの最大値を返す。

```
bddcost BDDCT::Cost(const int ix) const
```

自分自身が記録している第 ix 番目の変数のコストを返す。変数レベルの最大値を n とすると、ix 番目の変数のレベルは (n-ix) である。ix が 0 以上 n 未満のときは記録しているコスト値を返す。ix が負の場合は 1 を返す。ix が n 以上の場合は bddcost\_null を返す。

```
bddcost BDDCT::CostOfLev(const int lev) const
```

自分自身が記録している lev 番目のレベルの変数のコストを返す。変数レベルの最大値を n とすると、lev が 0 以下の場合は bddcost\_null を返す。Lev が n より大きい場合は 1 を返す。

```
char* BDDCT::Label(const int ix) const
```

自分自身が記録している第 ix 番目の変数のラベル文字列を返す。変数レベルの最大値を n

とすると、 $ix$  番目の変数のレベルは  $(n-ix)$  である。 $ix$  が負または  $n$  以上の場合は 0 を返す。

```
char* BDDCT::LabelOfLev(const int lev) const
```

自分自身が記録している  $lev$  番目のレベルの変数のラベル文字列を返す。変数レベルの最大値を  $n$  とすると、 $lev$  が 0 以下または  $n$  より大きい場合は 0 を返す。

```
int BDDCT::Alloc(const int n, const bddcost cost = 1)
```

$n$  個の変数のコストを記憶する領域を確保して、全ての変数のコストの初期値を  $cost$  とする。演算結果テーブルを初期化する。通常は 0 を返すが、領域確保に失敗した場合は異常終了して 1 を返す。

```
int BDDCT::Import(FILE* fp = stdin)
```

$fp$  で指定したファイルからデータを読み込んで、必要なサイズの表を確保してコスト値やラベルを記録する。正常終了の場合は 0 を返し、異常終了の場合は 1 を返す。以下にファイルフォーマットの例を示す。1 行目は変数の個数を表す。2 行目以降は各変数のコスト値を表す。各行の # より後、次の空白または改行まではラベル文字列を表す。ラベル文字列は省略してもよい。

```
#n 5
123 #lev5
456 #lev4
789 #lev3
-987 #lev2
-321 #lev1
```

```
int BDDCT::Rand(const int n, const bddcost min, const bddcost max)
```

$n$  個の変数のコストを記憶する領域を確保して、各変数にランダムなコスト値を設定する。乱数値の範囲は  $min$  以上  $max$  以下である。演算結果テーブルを初期化する。通常は 0 を返すが、領域確保に失敗した場合は異常終了して 1 を返す。

```
int BDDCT::SetCost(const int ix, const bddcost cost)
```

自分自身の第  $ix$  番目の変数にコスト値  $cost$  を代入する。変数レベルの最大値を  $n$  とすると、 $ix$  番目の変数のレベルは  $(n-ix)$  である。 $ix$  が 0 以上  $n$  未満でなければならない。演算結果テーブルを初期化する。正常終了時は 0 を返す。範囲外の  $ix$  を与えた場合や領域確保に失敗した場合は異常終了して 1 を返す。

`int BDDCT::SetCostOfLev(const int lev, const bddcost cost)`

自分自身の lev 番目のレベルの変数にコスト値 cost を代入する。変数レベルの最大値を n とすると、lev は 1 以上 n 以下でなければならない。演算結果テーブルを初期化する。正常終了時は 0 を返す。範囲外の lev を与えた場合や領域確保に失敗した場合は異常終了して 1 を返す。

`int BDDCT::SetLabel(const int ix, const char* label)`

自分自身の第 ix 番目の変数にラベル文字列 label を代入する。文字列長が BCT\_LabelLen を超える場合は、先頭の BCT\_LabelLen 文字のみを代入する。変数レベルの最大値を n とすると、ix 番目の変数のレベルは (n-ix) である。ix が 0 以上 n 未満でなければならない。正常終了時は 0 を返す。範囲外の ix を与えた場合は異常終了して 1 を返す。

`int BDDCT::SetCostOfLev(const int lev, const bddcost cost)`

自分自身の lev 番目のレベルの変数にラベル文字列 label を代入する。文字列長が BCT\_LabelLen を超える場合は、先頭の BCT\_LabelLen 文字のみを代入する。変数レベルの最大値を n とすると、lev は 1 以上 n 以下でなければならない。正常終了時は 0 を返す。範囲外の lev を与えた場合は異常終了して 1 を返す。

`Void BDDCT::Export(void) const`

自分自身の内容を標準出力ファイルに出力する。データフォーマットは Import と同じである。

`bddcost BDDCT::CacheRef(const unsigned char op, const bddword id) const`

演算種類 op と BDD または ZBDD の ID キーとして演算結果テーブルを参照し、ヒットすれば値を返す。ヒットしなければ bddcost\_null を返す。

`int BDDCT::CacheEnt(const unsigned char op, const bddword id, const bddcost cost)`

演算種類 op と BDD または ZBDD の ID キーとして演算結果テーブルに cost 登録する。正常終了した場合は 0 を返す。メモリが不足した場合は異常終了し 1 を返す。

`int BDDCT::CacheClear(void)`

演算結果テーブルを初期化する。正常終了した場合は 0 を返す。異常終了した場合は 1 を返す。

`int BDDCT::CacheEnlarge(void)`

演算結果テーブルのサイズを拡大する。正常終了した場合は 0 を返す。メモリ不足で異常

終了した場合は 1 を返す。

`bddcost BDDCT::MinCost(count ZBDD& f)`

ZBDD `f` に含まれる入力組合せ（解集合）の中でコスト最小の値を返す。`f` が空集合の場合は `bddcost_null` を返す。`f` のすべてのサブグラフに対する部分計算結果を演算結果テーブルに残す。

`bddcost BDDCT::MaxCost(count ZBDD& f)`

ZBDD `f` に含まれる入力組合せ（解集合）の中でコスト最大の値を返す。`f` が空集合の場合は `bddcost_null` を返す。`f` のすべてのサブグラフに対する部分計算結果を演算結果テーブルに残す。

`bddcost BDDCT::MinCost(count ZBDD& f)`

ZBDD `f` に含まれる入力組合せ（解集合）の中でコスト最小の値を返す。`f` が空集合の場合は `bddcost_null` を返す。

`ZBDD BDDCT::CostLE(count ZBDD& f, const bddcost bound)`

ZBDD `f` に含まれる入力組合せ（解集合）の中で、コストが `bound` 以下である解のみを集めた組合せ集合を表す ZBDD を生成し、それを返す。メモリあふれの場合は ZBDD (-1) を返す。