iZ-C Reference Manual

version 3.7

NTT DATA SEKISUI SYSTEMS

2020 年 11 月 20 日

Contents:

第 1 章

はじめに

iZ は効率的で拡張性のある制約解消ライブラリです。iZ は資源割り当て、スケジューリング、プランニング、生産制御のような複雑な組み合わせ論的問題を宣言的に表現し、解決することを目的としています。本版では、iZ のうち C 言語用のライブラリ iZ-C について記述します。このマニュアルは iZ-C のリファレンスであり、iZ-C で使うことのできるすべての関数を記述しています。

なお、iZ-C は以下の URL より入手可能です。

http://www.constraint.org/ja/izc_download.html

第 2 章

実行環境

2.1 開始と終了

iZ-C の各種 API 関数を呼び出す前に *cs_init()* を呼び出す必要があります。また、プログラムの終了時には *cs_end()* を呼び出す必要があります。

void **cs_init** (void)
: iZ-C が利用するメモリ領域をセットアップします。

void **cs_end** (void)
: *cs_init()* が呼ばれてから cs_end() が呼ばれるまでに割り付けられたメモリ領域 (それは CSint 型変数の作成や制約の宣言の時などに自動的に割り付けられたものです) は、cs_end() が呼ばれたときにすべて解放されます。

  cs_end() の後に CSint 型変数にアクセスすることはできません。

2.2 エラーの取得

ライブラリ内部には、エラーが発生したことを記録する変数が存在し、この変数を読み出すことによってエラーの発生を知ることができます。(以降この変数を err と呼びます)

err は API を呼び出すことで、読み出しあるいは設定が可能です。

void **cs_initErr** (void)
: cs_initErr() を呼び出すと err が CS_ERR_NONE に初期化されます。cs_initErr() は *cs_init()* の中で呼び出されています。

int **cs_getErr** (void)
: 現在の err の値を返します。*CS_ERR_NONE* が返るということは, 前回の cs_getErr() あるいは *cs_init()* の呼び出し以降、今回の cs_getErr() の呼び出しまでの間にはエラーは発生していないということを意味します。

なお、err の値は以下の整数値をとります。

CS_ERR_NONE
エラーは発生していない

CS_ERR_GETVALUE
即値化されていない変数に対して *cs_getValue()* を呼び出した

CS_ERR_OVERFLOW
計算中にオーバーフローが発生した

CS_ERR_NO_MEMORY
メモリ取得に失敗した

void **cs_setErr** (int *code*)
err の値を *code* として設定し直します。

2.3 エラー発生時の挙動の設定

エラーが発生した際に実施するアクションを、あらかじめ関数として登録しておくことができます。登録された関数はエラーが発生したときにエラー内容を err に設定した後で呼び出されます。

void **cs_setErrHandler** (int *code*, void (**func*)(void* data, void* ext), void* *ext*)

code のエラーが発生したときに呼び出される関数を *func* として設定します。*func* の引数 *data* は、*code* によって異なる意味を持ちます。*func* の引数 *ext* は、cs_setErrHandler の呼び出し時に与えられた *ext* と同じものです。

code の値	data の値	デフォルトの挙動
CS_ERR_GETVALUE	CSint へのポインタ	メッセージを出力して処理を続行
CS_ERR_OVERFLOW	NULL	何もしないで続行
CS_ERR_NO_MEMORY	NULL	abort() を呼び出す (プログラムの実行は終了)

2.4 バージョン情報の取得

const char* **cs_getVersion** (void)
ライブラリのバージョンを現す文字列を取得します。(例: "3.5.0")

以下の C プリプロセッサシンボルを文字列化して "." で連結したものと等価です。

IZ_VERSION_MAJOR
iZ-C のメジャーバージョンを表す数値。

IZ_VERSION_MINOR
iZ-C のマイナーバージョンを表す数値。

IZ_VERSION_PATCH
iZ-C のパッチレベルを表す数値。

2.5 統計情報の出力

void **cs_printStats** (void)
以下の 3 種類の情報を出力します。

- Nb Fails (解探索の過程で発生したフェイルの数)
- Nb Choice Points (バックトラックすることが可能な変数の即値化の回数)
- Heap Size (バックトラックができるようにコンテキストを保存しているヒープ領域のサイズ)

Nb Fails および Nb Choice Points はそれぞれ、*cs_getNbFails()* および *cs_getNbChoicePoints()* 関数で直接取得できます。

void **cs_fprintStats** (FILE *f)
この関数は引数として出力先のファイルを示す FILE ポインタをとる以外は *cs_printStats()* と同じです。

第 3 章

領域変数と基本的な関数

CSint 型の変数は整数の領域変数です。定義済みの iZ-C の関数では、CSint へのポインタ (CSint*) のみが使用されます。領域変数はライブラリの管理下にあるため、作成した領域変数を利用者が解放することはできません。

3.1 コンストラクタ

CSint 型の領域変数は、以下の関数によって構築することができます。

CSint ***cs_createCSint** (int *min*, int *max*)
{ *min* .. *max* } を領域とする CSint 型変数を作成します。

CSint ***cs_createNamedCSint** (int *min*, int *max*, char **name*)
{ *min* .. *max* } を領域とする CSint 型変数を作成し、名前をつけます。(cs_createCSint() の後に cs_setName() を用いるのと同じです。)

CSint ***CSINT** (int *n*)
すでに即値化された（つまり、その領域の要素が１つしかない）CSint 型変数を作成するのに使います。この関数は、cs_createCSint(*n* , *n*) と等価です。

CSint ***cs_createCSintFromDomain** (int **array*, int *size*)
size の大きさをもつ *array* により定義される領域を持つ CSint 型変数を作成します。

CSint ****cs_createCSintArray** (int *nbVars*, int *min*, int *max*)
nbVars 個の CSint 型変数よりなる配列を作成します。各変数の領域は、すべて { *min* .. *max* }になります。

<注意> 本 API の返り値は領域変数の実装である CSint へのポインタの配列です。ポインタを格納した配列も制約処理系の管理下となるため、作成した配列を利用者が解放することはできません。

3.2 領域変数の情報にアクセスするための関数

CSint 型の領域変数に関する情報にアクセスするためには、以下の基本的な関数を使用します。

int **cs_getMin** (CSint *vint*)

vint の領域の最小値を返します。

int **cs_getMax** (CSint *vint*)

vint の領域の最大値を返します。

int **cs_getNbElements** (CSint *vint*)

vint の領域に含まれる要素数を返します。

int **cs_getNbConstraints** (CSint *vint*)

vint に設定されている制約の数を返します。設定されたときに失敗した制約は数に含まれません。

char ***cs_getName** (CSint *vint*)

cs_setName() を用いて *vint* に設定された名前を返します。

CSint 型の変数の名前は、*cs_printf()* や *cs_fprintf()* でフォーマット記述子 %T を使った場合にも表示されます。

int **cs_getNextValue** (CSint *vint*, int *val*)

vint の領域に含まれる値の集合の中で、*val* よりも大きい最初の要素を返します。

val が cs_getMax(*vint*) で返る値より大きい場合には、本関数は INT_MAX (int 型の最大値) を返します。

以下の例では、cs_getNextValue() は for 文中で使われています。:

```
void display(CSint *vint)
{
  int val;
  for (val = cs_getMin(vint); val <= cs_getMax(vint); val = cs_getNextValue(vint,␣
↪val))
      printf("%d ", val);
}
```

上記の関数 display() は CSint 型の変数のとりうる値（=領域に含まれる値）をすべて、標準出力に昇順に出力します。

int **cs_getPreviousValue** (CSint *vint*, int *val*)

vint の領域に含まれる値の集合の中で、*val* よりも小さい最初の要素を返します。

val が cs_getMin(*vint*) で返る値より小さい場合には、本関数は INT_MIN (int 型の最小値) を返します。

int ***cs_getDomain** (CSint *vint*)

vint の領域の要素を長さ cs_getNbElements(vint) の配列の各要素の値として返します。返された配列のメモリは不要になったら cs_freeDomain で解放する必要があります。

void **cs_freeDomain** (int* *array*)

cs_getDomain() で取得した配列のメモリを解放します。

IZBOOL char **cs_isIn** (CSint *vint*, int *val*)

val が *vint* の領域の要素であれば TRUE を返します。そうでなければ FALSE を返します。

void **cs_setName** (CSint *vint*, char *name)

CSint 型変数に名前を付けるために使います。CSint 型変数が表示されるとき (*cs_printf()* あるいは *cs_fprintf()* を使用します)、付けられた名前も一緒に表示されます。(書式指定子 %T が指定された場合)

IZBOOL **cs_isFree** (CSint *vint)

vint がまだ即値化されていなければ TRUE を返します。 そうでなければ FALSE を返します。

「即値かされていない」とはすなわち、cs_getNbElements(*vint*) > 1 あるいは cs_getMin(*vint*) < cs_getMax(*vint*) であるような状態です。

IZBOOL char **cs_isInstantiated** (CSint *vint)

vint が即値化されていれば TRUE を返します 。そうでなければ FALSE を返します。

「即値かされている」とはすなわち、cs_getNbElements(*vint*) == 1 あるいは cs_getMin(*vint*) == cs_getMax(*vint*) であるような状態です。

int **cs_getValue** (CSint *vint)

CSint 変数 *vint* が即値化されていれば、その値を返します。*vint* がまだ即値化されていない場合にはエラーが発生し (cs_getErr() を呼び出すと CS_ERR_GETVALUE が返ります。)、cs_getValue(*vint*) は cs_getMin(*vint*) を返します。(デフォルトの動作の場合)

void **cs_printf** (const char *control, ...)

この関数は、CSint 型の変数および CSint 型の変数の配列用の変換指定が 3 種類追加された以外は、C 言語で標準の printf() 関数と同じです。

- `cs_printf("%T", vint)` は *vint* の名前を (名前があれば) 表示し、ついでその領域を標準出力に出力します。
- `cs_printf("%D", vint)` は *vint* の領域のみを標準出力に出力します。
- `cs_printf("%A", array, size)` は *size* 個の要素よりなる CSint 型変数の配列 *array* に含まれる各変数を表示します。もし名前があれば名前も表示します。変数間はコンマで区切られます。

void **cs_fprintf** (FILE *f, const char *control, ...)

この関数は引数として出力先のファイルを示す FILE ポインタ *f* をとる以外は cs_printf() と同じです。

3.3 領域に関する制約

以下の関数は制約であり、設定した時点でフェイルする (つまり、FALSE を返す) ことがあります。

IZBOOL **cs_InArray** (CSint *vint*, int *array*, int *size*)

CSint 型変数 *vint* が長さ *size* の整数の配列 *array* に含まれる値を領域として持つように制約します。

IZBOOL **cs_NotInArray** (CSint **vint*, int **array*, int *size*)

CSint 型変数 *vint* が *array* 配列に含まれる値を領域として持たないように制約します。*array* に含まれるすべての値は *vint* の領域から取り除かれます。

これは以下のように定義できるでしょう。:

```
IZBOOL cs_notInArray(CSint *vint, int *array, int size) {
  int i;
  for (i = 0; i < size; i++)
   if (!cs_NEQ(vint, array[i])
     return FALSE;

  return TRUE;
}
```

IZBOOL **cs_InInterval** (CSint **vint*, int *min*, int *max*)

CSint 型変数 *vint* が{ *min* .. *max* }に含まれるような領域を持つように制約します。(つまり cs_getMin(*vint*) >= *min* かつ cs_getMax(*vint*) <= *max* が成立する。)

これは以下のように定義できるでしょう。:

```
IZBOOL cs_InInterval(CSint *vint, int min, int max) {
  return(cs_GE(vint, min) && cs_LE(vint, max));
}
```

IZBOOL **cs_NotInInterval** (CSint **vint*, int *min*, int *max*)

CSint 型変数 *vint* が区間{ *min* .. *max* }に含まれる値を領域として持たないように制約します。*vint* の領域で区間{ *min* .. *max* }に含まれる値はすべて取り除かれます。

これは以下のように定義できるでしょう。:

```
IZBOOL cs_NotInInterval(CSint *vint, int min, int max) {
  int i;
  for (i = min; i <= max; i++)
    if (!cs_NEQ(vint, i))
      return FALSE;

  return TRUE;
}
```

第 4 章

算術制約

以下の制約では、すでに存在する CSint 型変数間の算術的な関係により定義される新しい CSint 型変数が作成されます。これらの制約は、設定されたときにはフェイルしません。

CSint ***cs_Add*** (CSint **vint1*, CSint **vint2*)
その領域が *vint1* と *vint2* の和と等しい CSint 型変数を返します。

CSint ***cs_VAdd*** (int *nbVars*, CSint **vint*, ...)
cs_VAdd() は可変長引数を受け付けます (引数として与えられる CSint 型変数の数は第 1 引数 nbVars で指定されます。)

引数として与えられた CSint 型変数の和であるような CSint 型変数を返します。

`cs_VAdd(n, V1, V2, ..., Vn)` = V1 + V2 + ... + Vn

CSint ***cs_Sub*** (CSint **vint1*, CSint **vint2*)
その領域が *vint1* から *vint2* を引いた値に等しい CSint 型変数を返します。

<注意!> この制約は、関数名の通り、返される CSint 型変数を *v* とすると、 *vint1* = *v* + *vint2* と等しくなります。*vint1* と *vint2* は可換ではなく、*v* は *vint1* と *vint2* の差（絶対値）ではありません。

CSint ***cs_VSub*** (int *nbVars*, CSint **vint*, ...)
cs_VSub() は可変長引数を受け付けます (引数として与えられる CSint 型変数の数は第 1 引数 nbVars で指定されます。)

引数として与えられた CSint 型変数 V1, V2, V3, ... , Vn について、V1 - V2 - V3 - ... - Vn であるような CSint 型変数を返します。

`cs_VSub(n, V1, V2, V3, ..., Vn)` = V1 - V2 - V3 - ... - Vn

CSint ***cs_Mul*** (CSint **vint1*, CSint **vint2*)
その領域が *vint1* と *vint2* の積と等しい CSint 型変数を返します。

CSint ***cs_VMul*** (int *nbVars*, CSint **vint*, ...)
cs_VMul() は可変長引数を受け付けます (引数として与えられる CSint 型変数の数は第 1 引数 *nbVars* で指

定されます。)

引数として与えられた CSint 型変数の積である CSint 型変数を返します。

`cs_VMul(n, V1, V2, ..., Vn)` = V1 * V2 * ... * Vn

CSint *__cs_Div__ (CSint *vint1*, CSint *vint2*)

vint1 を vint2 で割った商になるような CSint 型変数を返します。

<注意!>この制約は、返される CSint 型変数を v とすると、*vint1 = v * vint2* として実装されています。整数間の割り算（であって有理数ではない）のため、実装の方法の違いは結果の違いになりえます。また、*vint2* が 0 の場合にも本関数はフェイルしません。このとき v は (INTMIN..INTMAX) の領域を持つものとして作成されます。

CSint *__cs_VDiv__ (int *nbVars*, CSint *vint, ...)

cs_VDiv() は可変長引数を受け付けます (引数として与えられる CSint 型変数の数は第 1 引数 nbVars で指定されます。)

CSint *__cs_Sigma__ (CSint **array, int *size*)

要素数が *size* の CSint 型変数の配列 *array* の和に等しい CSint 型変数を返します。.

CSint *__cs_ScalProd__ (CSint **array, int *vector, int *size*)

要素数が *size* の CSint 型変数のベクトル *array* と同じ要素数の整数のベクトル *vector* のスカラ積に等しい CSint 型変数を返します。

CSint *__cs_VScalProd__ (int *nbVars*, CSint *vint, ...)

cs_VScalProd() は可変長引数を受け付けます (CSin 型変数および整数の数は第 1 引数 nbVars で指定されます)。

cs_VScalProd(n, V1, V2, ..., Vn, c1, c2, ..., cn) は CSint 型変数のベクトル (V1, V2, ..., Vn) と整数のベクトル (c1, c1, ..., cn) のスカラ積を返します。

CSint *__cs_Abs__ (CSint *vint*)

CSint 型変数 *vint* の絶対値と等しい CSint 型変数を返します。

第 5 章

関係制約

以下の関数はすでに存在する CSint 変数間に制約を設定します。これらの関数は呼び出し時に直ちにフェイルする (つまり FALSE を返す) ことがあります。

IZBOOL **cs_Le** (CSint **vint1*, CSint **vint2*)
: *vint1* は *vint2* より小さいか等しくなければなりません。(つまり cs_getMin(*vint1*) <= cs_getMin(*vint2*) でかつ cs_getMax(*vint1*) <= cs_getMax(*vint2*)).

IZBOOL **cs_Ge** (CSint **vint1*, CSint **vint2*)
: *vint1* は *vint2* より大きいか等しくなくてはなりません。

IZBOOL **cs_Lt** (CSint **vint1*, CSint **vint2*)
: *vint1* は *vint2* より小さくなければなりません。

IZBOOL **cs_Gt** (CSint **vint1*, CSint **vint2*)
: *vint1* は *vint2* より大きくなくてはなりません。

IZBOOL **cs_Eq** (CSint **vint1*, CSint **vint2*)
: *vint1* と *vint2* は常に等しくなくてはなりません。

IZBOOL **cs_Neq** (CSint **vint1*, CSint **vint2*)
: *vint1* と *vint2* は異なった値を持たねばなりません。

以下の関数は、上述の対応する各関数における 2 つのオペランドのうち、一方が定数の場合に用いると便利です。

- `IZBOOL cs_LE(CSint *vint, int val)`
- `IZBOOL cs_GE(CSint *vint, int val)`
- `IZBOOL cs_LT(CSint *vint, int val)`
- `IZBOOL cs_GT(CSint *vint, int val)`
- `IZBOOL cs_EQ(CSint *vint, int val)`
- `IZBOOL cs_NEQ(CSint *vint, int val)`

例えば、cs_LE() は以下のように定義できるでしょう。:

```
IZBOOL cs_LE(CSint *vint, int val) {
  return(cs_Le(vint, CSINT(val));
}
```

<注意!>しかしながら、この例は実装を反映したものではありません。一方が定数であるような制約は、ある領域変数に対する単項制約と考えることもでき、この場合、2 つの領域変数に対する 2 項の関係制約とは異なるクラスに属する制約と考えられます。iZ-C でも、実装上は、この考え方に基づいています。従って、例えば cs_LE() が呼ばれた場合には、第 1 引数の領域変数が 第 2 引数の値以下の領域を持つかどうかの検査を行うのであって、第 2 引数の値で即値化された領域変数を作成し、第 1 引数の領域変数との間に制約を設定するわけではありません。メモリの利用効率の面から、単項制約の方が有利であるのは明らかです。

IZBOOL **cs_AllNeq**(CSint ***array*, int *size*)

array に属する CSint 変数はすべて異なる値を持たねばなりません。この制約は、以下のように書くこともできるでしょう。:

```
IZBOOL cs_AllNeq(CSint **array, int size) {
  int i, j;
  for (i = 0; i < size - 1; i++)
    for (j = i + 1; j < size; j++)
      cs_Neq(array[i], array[j]);
}
```

＜注意！＞この場合も実装上は、上記は等価ではありません。cs_AllNeq() は、複数の項に対する N-ary 制約として実装されており、2 項制約の組み合わせで実装されているのではありません。この場合も、メモリの利用効率・制約伝播の点で、N-ary 制約としての実装が優れているのは明らかです。なお、iZ-C での N-ary 制約のクラスに属する他の制約としては、後述の when 制約（デモン）があります。

以下の制約では、すでに存在する CSint 型変数間の関係が成立するとき 1、成立しないとき 0 という値をとるような新しい CSint 型変数が作成されます。これらの制約は、設定されたときにはフェイルしません。

CSInt* **cs_ReifLe**(CSint ***vint1*, CSint ***vint2*)

vint1 が *vint2* より小さいか等しいとき 1、そうでないときは 0 となるような CSint 型変数を返します。

CSInt* **cs_ReifGe**(CSint ***vint1*, CSint ***vint2*)

vint1 が *vint2* より大きいか等しいとき 1、そうでないときは 0 となるような CSint 型変数を返します。

CSInt* **cs_ReifLt**(CSint ***vint1*, CSint ***vint2*)

vint1 が *vint2* より小さいとき 1、そうでないときは 0 となるような CSint 型変数を返します。

CSInt* **cs_ReifGt**(CSint ***vint1*, CSint ***vint2*)

vint1 が *vint2* より大きいとき 1、そうでないときは 0 となるような CSint 型変数を返します。

CSInt* **cs_ReifEq**(CSint ***vint1*, CSint ***vint2*)

vint1 と *vint2* が等しいとき 1、そうでないときは 0 となるような CSint 型変数を返します。

CSInt* **cs_ReifNeq**(CSint **vint1*, CSint **vint2*)

vint1 と *vint2* が異なるとき 1、そうでないときは 0 となるような CSint 型変数を返します。

以下の関数は、上述の対応する各関数における 2 つのオペランドのうち、一方が定数の場合に用いると便利です。

- `CSint* cs_ReifLE(CSint *vint, int val)`
- `CSint* cs_ReifGE(CSint *vint, int val)`
- `CSint* cs_ReifLT(CSint *vint, int val)`
- `CSint* cs_ReifGT(CSint *vint, int val)`
- `CSint* cs_ReifEQ(CSint *vint, int val)`
- `CSint* cs_ReifNEQ(CSint *vint, int val)`

第 6 章

高水準制約

以下の制約は、今までのものに比べてより複雑なものです。これらは、これまでに述べてきたプリミティブとデモン (デモン 参照) により実現することができます。

<注意!>実際の iZ-C における実装では、以下の制約は既述のプリミティブとデモンの組み合わせによるのではなく、独自の実現方式を持ちます。

特定の関係を満たす CSint を返す制約では、制約設定時にフェイルした場合には NULL が返されます。

IZBOOL **cs_IfEq** (CSint **vint1*, CSint **vint2*, int *val1*, int *val2*)
vint1 が値 *val1* に即値化されたら、*vint2* を *val2* に即値化します。

vint2 が値 *val2* に即値化されたら、*vint1* を *val1* に即値化します。

IZBOOL **cs_IfNeq** (CSint **vint1*, CSint **vint2*, int *val1*, int *val2*)
CSint 型変数の組 (*vint1*, *vint2*) は値 (*val1*, *val2*) を持つことができません。

CSint ***cs_Occur** (CSint **vint*, int *val*, CSint ***array*, int *size*)
値 *val* は *vint* 回 *array* 中に出現しなくてはなりません。

<注意!>旧バージョンでは実装に不備がありました。本来ならば返り値は値 *val* が *vint* 回 *array* 中に出現する (TRUE) かしない (FALSE) かを返す必要がありました。

旧バージョンとの互換性のため返り値は変更しませんが、今後は *cs_OccurConstraints()* を使ってください。本 API は将来削除する予定です。

CSint ***cs_OccurDomain** (int *val*, CSint ***array*, int *size*)
array 中に値 *val* となりうる個数の範囲を返します。

IZBOOL **cs_OccurConstraints** (CSint **vint*, int *val*, CSint ***array*, int *size*)
値 *val* は *vint* 回 *array* 中に出現しなくてはなりません。

CSint ***cs_Index** (CSint ***array*, int *size*, int *val*)
返される CSint 型変数を *Index* とすると、長さ *size* の配列 *array* に対して要素制約 *array* [*Index*] = *val* が設定されます。

i ∈ domain(*Index*) ⇔ *val* ∈ domain(*array* [*i*])

CSint ***cs_Element*** (CSint **index*, int **values*, int *size*)
返される CSint 型変数を *Element* とすると、長さ *size* の配列 *values* について、*Element* = *values* [*index*] が設定されます。

CSint ***cs_VarElement*** (CSint **index*, CSint ***array*, int *size*)
返される CSint 型変数を *Element* とすると、長さ *size* の領域変数の配列 *array* について、*Element* = *array* [*index*] が設定されます。

CSint ***cs_VarElementRange*** (CSint **index*, CSint ***array*, int *size*)
CSint 型変数を *cs_VarElement()* と同様に制約します。ただし、*array* 内の変数については上限と下限のみに基づいて制約伝播を行います。

CSint ***cs_Min*** (CSint ***array*, int *size*)
array により参照される CSint 型変数のうち最小のものの値と等しい CSint 型変数を返します。

CSint ***cs_VMin*** (int *nbVars*, CSint **vint*, ...)
cs_VMin() は可変長引数をとります (引数の数は最初の引数 *nbVars* に指定されます)。

CSint ***cs_Max*** (CSint ***array*, int *size*)
array により参照される CSint 変数のうち最大のものの値と等しい CSint 型変数を返します。

CSint ***cs_VMax*** (int *nbVars*, CSint **vint*, ...)
cs_VMax() は可変長引数をとります (引数の数は最初の引数 *nbVars* に指定されます)。

IZBOOL ***cs_Cumulative*** (CSint** *startVars*, CSint** *durationVars*, CSint** *resourceVars*, int *size*, CSint* *limitVar*)
スケジューリング等で使用される制約です。*size* 個のタスクの i 番目について、開始時刻を *startVars* [i]、継続時間を *durationVars* [i]、必要なリソースを *resourceVars* [i] としたとき、同時に実行されるタスクの *resourceVars* の合計が *limitVar* 以下になるように制約します。

IZBOOL ***cs_Disjunctive*** (CSint** *startVars*, CSint** *durationVars*, int *size*)
スケジューリング等で使用される制約です。*size* 個のタスクの i 番目について、開始時刻を *startVars* [i]、継続時間を *durationVars* [i] としたとき、タスクが同一時刻に実行されないように制約します。

IZBOOL ***cs_Regular*** (CSint** *array*, int *size*, const int* *d*, int *Q*, int *S*, int *q0*, const int* *F*, int *fsize*)
array により参照される CSint 型変数配列を、*d*, *Q*, *S*, *q0*, *F* で定義されるオートマトンで受理される記号列となるように制約します。

- *d* は $Q \times S$ の大きさを持つ配列です。状態 q で入力 s を受け取った時の次の遷移先が位置 $q \times S + s$ に格納されている必要があります。(遷移先がない場合は、-1)
- *Q* は状態数です。
- *S* は入力の種類数です。
- *q0* は初期状態です。

- *F* は受理状態を格納した配列で、その長さは *fsize* です。

第 7 章

解探索とヒューリスティクス

iZ-C が提供する解探索・生成メカニズムの基本原理は、以下の通りです。

1. まだ即値化されていない CSint 型変数 *var* を、CSint 型変数の集合 *allvars* から 1 つ見つけます。即値化されていない変数が見つからなければ (全ての変数が即値化されていれば)、探索は終了です。
2. 見つけた変数 *var* の領域を縮小します。例えば、値 *val* を選んで、以下のようにしてこの値で即値化してみます。: `cs_EQ(var, val)`
   - 領域の縮小に失敗した場合は、ステップ 2 に戻って別の縮小方法を試します。例えば、別の値 *val2* を選択します。
   - 即値化に成功した場合にはステップ 1 に行きます。

7.1 基本的な解探索

IZBOOL **`cs_search`** (CSint ***allvars***, int *nbVars*, CSint* (***findFreeVar***)(CSint **allvars, int nbVars))

この関数は、*allvars* から参照可能なすべての CSint 型変数を即値化しようとします。解がなければ FALSE を返します。*findFreeVar* 関数は *allvars* の中から即値化されていない CSint 型変数を 1 つ見つけて返すための関数です。

IZBOOL **`cs_Vsearch`** (int *nbVars*, CSint ***vint***, ...)

cs_Vsearch() は可変長引数をうけつけます (引数として渡される変数の数は第 1 引数 *nbVars* で指定されるものとします)。

7.2 組み込みの選択関数

CSint ***`cs_findFreeVar`** (CSint ***allvars***, int *nbVars*)

allvars 中の CSint 型変数で、即値化されていないもののうち、最初に見つかった (=添字番号が小さい) ものを返します。

CSint ***cs_findFreeVarNbElements** (CSint ***allvars*, int *nbVars*)

即値化されていない CSint 型変数のうち、領域の要素数が最も少ないものを返します。そのような変数が複数ある場合には、最初に見つかった (= 添字番号が小さい) ものを返します。

CSint ***cs_findFreeVarNbElementsMin** (CSint ***allvars*, int *nbVars*)

即値化されていない CSint 型変数のうち、領域の要素数が最も少ないものを返します。そのような変数が複数ある場合には、領域の最小値が最も小さいものを返します。

CSint ***cs_findFreeVarNbConstraints** (CSint ***allvars*, int *nbVars*)

即値化されていない CSint 型変数のうち、設定されている制約の数が最も多いものを返します。

例えば、cs_findFreeVarNbElements() は以下のように定義できるでしょう。:

```
CSint *cs_findFreeVarNbElements(CSint **allvars, int nbVars) {
  int i;
  int nbElements, nbElementsOpt;
  CSint *varOpt = NULL;

  nbElementsOpt = INT_MAX;
  for (i = 0; i < nbVars; i++) {
    nbElements = cs_getNbElements(allvars[i]);
    if ((nbElements > 1) && (nbElements < nbElementsOpt)) {
      nbElementsOpt = nbElements;
      varOpt = allvars[i];
    }
  }
  return(varOpt);
}
```

7.3 フェイル数とチョイスポイント数

以下の関数を用いれば、フェイル数やチョイスポイント数を用いたヒューリスティクスを書くこともできます。

int **cs_getNbFails** ()

この関数が呼ばれた時点までに発生したフェイルの数を返します。フェイルを発生させる関数としては、以下があります。

- *cs_search()*
- *cs_searchFail()*
- *cs_Vsearch()*
- *cs_searchCriteria()*
- *cs_searchCriteriaFail()*

- *cs_searchValueSelectorFail()*
- *cs_searchValueSelectorRestartNG()*
- *cs_searchMatrix()*
- *cs_searchMatrixFail()*
- *cs_findAll()*
- *cs_minimize()*

int **cs_getNbChoicePoints**()

この関数が呼ばれた時点までに発生したチョイスポイントの数を返します。チョイスポイントを発生させる関数は、*cs_getNbFails()* と同じです。

7.4 複雑な解探索関数

IZBOOL **cs_searchCriteria**(CSint **allvars, int nbVars, int (*findFreeVar)(CSint **allvars, int nbVars), int (*criteria)(int index, int val))

cs_search() では CSint 型変数 *var* が選ばれると、*var* の領域に含まれない値を飛ばしながら値を昇順に (cs_getMin(*var*) から cs_getMax(*var*) へと) 試していきます。

cs_searchCriteria() を使えば、この選択の順序を制御することができます。*criteria()* 関数の返り値が最小になる値が最初に選ばれます。

IZBOOL **cs_searchMatrix**(CSint ***matrix, int NbRows, int NbCols, int (*findFreeRow)(CSint ***matrix, int NbRows, int NbCols), int (*findFreeCol)(int row, CSint **Row, int NbCols), int (*criteria)(int row, int col, int val))

cs_search() と *cs_searchCriteria()* は CSint 型変数のベクトルについて解を生成するために使われます。

cs_searchMatrix() はマトリックスに対して同じ目的で使われます。*findFreeRow()* は選択したい行のインデックスを返します。行が選ばれると *findFreeCol()* が選択された列のインデックスを返します。CSint 型変数 (Col, Row) が選択されると *criteria()* 関数の返り値が最小になる値が選ばれます。

IZBOOL **cs_searchFail**(CSint **allvars, int nbVars, CSint* (*findFreeVar)(CSint **allvars, int nbVars), int NbFailsMax)

cs_searchFail() はバックトラックの回数が *NbFailsMax* に到達したら停止し FALSE を返す (つまりフェイルする) ことを除けば *cs_search()* と同じです。

NbFails を調べれば、フェイルが発生した理由がバックトラックの制限によるものかどうかを知ることができます。(*cs_getNbFails()* 関数を参照。)

IZBOOL **cs_searchCriteriaFail**(CSint **allvars, int nbVars, int (*findFreeVar)(CSint **allvars, int nbVars), int (*criteria)(int index, int val), int NbFailsMax)

cs_searchCriteriaFail() はバックトラックの回数が *NbFailsMax* に到達したら停止し FALSE を返

す (つまりフェイルする) ことを除けば *cs_searchCriteria()* と同じです。

NbFails を調べれば、フェイルが発生した理由がバックトラックの制限によるものかどうかを知ることができます。(*cs_getNbFails()* 関数を参照。)

IZBOOL **cs_searchValueSelectorFail**(CSint** *allvars*, const CSvalueSelector** *selectors*, int *nbVars*, int (**findFreeVar*)(CSint** allvars, int nbVars), int *NbFailsMax*, const CSsearchNotify* *notify*)

cs_searchValueSelectorFaill() は、*findFreeVar* で allvars から選択した変数に対して、selectors で指定された方法で領域を縮小して探索を行います。selectors は *cs_getValueSelector()* あるいは *cs_createValueSelector()* で生成される CSvalueSelector 型の変数へのポインタの配列で、探索対象の CSint 型変数 *var* の配列と同じサイズ (すなわち nbVars) である必要があります。

NbFailsMax の利用方法は *cs_searchFail()* と同じです。

notify は CSsearchNotify 型の変数へのポインタであり、ここにセットされたコールバック関数が探索の各フェーズで呼び出されます。詳細は *cs_createSearchNotify()* を参照してください。

IZBOOL **cs_searchValueSelectorRestartNG**(CSint** *allvars*, const CSvalueSelector** *selectors*, int *nbVars*, int (**findFreeVar*)(CSint** allvars, int nbVars), int (*maxFailFunction)(void* state), void* *maxFailState*, int *nbFailsMax*, CSnoGoodSet* *ngs*, const CSsearchNotify* *notify*)

cs_searchValueSelectorRestartNG() は、*cs_searchValueSelectorFail()* に対してリスタートおよび NoGood の機能を付加したものです。*maxFailFunction* が探索の開始時に呼び出され、許容されるフェイルの上限が決定されます。探索のフェイル数がこの上限に到達すると、全ての探索を再度実行し直します。*maxFailFunction* の呼び出し時には引数として *maxFailState* が与えられます。

探索の再実行では、再び同じフェイルが発生しないようにするために、フェイルが発生した条件 (NoGood) を *cs_createNoGoodSet()* で生成された変数 ngs に記録します。(実行速度やメモリの制限から記録される NoGood は発生したフェイルの一部分のみです)

ngs および *notify* の機能が不要な場合は NULL を渡すことができます。

典型的な呼び出し方は以下のようになります:

```
static int findFreeVarIndex(CSint **allvars, int nbVars) {
    for (int i = 0; i < nbVars; i++) {
        if (cs_isFree(allvars[i]))
            return i;
    }

    return -1;
}

static int nextFail(void* state) {
    int* pn = (int*)state;
```

(次のページに続く)

(前のページからの続き)

```
    int ret = *pn;

    *pn = ret + 10;

    return ret;
}

/* ... */
    const CSvalueSelector** vs = malloc(sizeof(CSvalueSelector*) * nbVars);
    CSnoGoodSet* ngs = cs_createNoGoodSet(allvars, nbVars, NULL, 0, NULL, NULL);
    CSsearchNotify* notify = cs_createSearchNotify(NULL);

    int fail = 10;

    for (int i = 0; i < nbVars; i++) {
        vs[i] = cs_getValueSelector(CS_VALUE_SELECTOR_MIN_TO_MAX);
    }

    cs_searchValueSelectorRestartNG(allvars, vs, nbVars, findFreeVarIndex,␣
↪nextFail, &fail, 100000, ngs, notify);
/* ... */
```

IZBOOL **cs_searchMatrixFail**(CSint ***matrix, int NbRows, int NbCols, int (*findFreeRow)(CSint ***matrix, int NbRows, int NbCols), int (*findFreeCol)(int row, CSint **Row, int NbCols), int (*criteria)(int row, int col, int val), int NbFailsMax)

cs_searchMatrixFail() はバックトラックの回数が *NbFailsMax* に到達したら停止し FALSE を返す (つまりフェイルする) ことを除けば *cs_searchMatrix()* と同じです。

NbFails を調べれば、フェイルが発生した理由がバックトラックの制限によるものかどうかを知ることができます。(*cs_getNbFails()* 関数を参照。)

IZBOOL **cs_findAll**(CSint **allvars, int nbVars, CSint* (*findFreeVar)(CSint **allvars, int nbVars), void (*found)(CSint **allvars, int nbVars))

可能な解をすべて探索します。解が 1 つ見つかるたびに *found()* 関数が呼ばれます。典型的なケースとして *found()* 関数は解を表示するのに使われます。:

```
void found(CSint **allvars, int nbVars) {
  static NbSolutions = 0;
  printf("Solution %d\n", ++NbSolutions);
  cs_printf("%A\n\n", allvars, nbVars);
}
```

IZBOOL **cs_minimize**(CSint **allvars, int nbVars, CSint* (*findFreeVar)(CSint **allvars, int nbVars), CSint *cost, void (*found)(CSint **allvars, int nbVars, CSint *cost))

cost を最小化する解を見つけようとします。 最小化のステップ毎に *found()* 関数が呼び出されます。典型的なケースとして、*found()* 関数はそのステップで得られた解を表示するのに使われます。:

```
void found(CSint **allvars, int nbVars, CSint *cost) {

  static NbSolutions = 0;

  printf("Solution %d\n", ++NbSolutions);
  cs_printf("%A\n", allvars, nbVars);
  cs_printf("Cost = %T\n\n", cost);
}
```

cs_minimize() は以下のように定義できます。:

```
IZBOOL cs_minimize(CSint **allvars, int nbVars, CSint* (*findFreeVar)(CSint␣
↪**allvars, int nbVars),  CSint *cost, void (*found)(CSint **allvars, int nbVars,
 CSint *cost)) {
  char gFirst, g;
  gFirst = g = cs_search(allvars, nbVars, findFreeVar);
  while (g) {
    int currentCost = getMin(cost);
    found(allvars, nbVars, cost);
    cs_restoreAll();
    g = (cs_LT(cost, currentCost) && cs_search(allvars, nbVars, findFreeVar));
  }
  return gFirst;
}
```

7.5 解探索関数の中断

void **cs_cancelSearch** (void)

現在実行中の探索を安全な位置で中断します。この関数は探索を実行しているスレッドとは別のスレッドから呼び出すことが可能です。必要があれば *cs_restoreContextUntil()* を使用してコンテキストを復帰してください。

7.6 領域縮小方法の指定

以下の探索関数では、CSvalueSelector 型の変数を引数に渡すことで、領域の縮小方法として即値化 (領域から値を 1 つ選択する) 以外の方法を指定することができます。

- *cs_searchValueSelectorFail()*
- *cs_searchValueSelectorRestartNG()*

7.6.1 組み込みの領域縮小方法の指定

const CSvalueSelector* **cs_getValueSelector** (int *vs*)

組み込みの領域縮小を実施するための CSvalueSelector を得ます。*vs* としては以下の値を指定できます。

CS_VALUE_SELECTOR_MIN_TO_MAX

領域の最も小さい値から順に試す。

CS_VALUE_SELECTOR_MAX_TO_MIN

領域の最も大きい値から順に試す。

CS_VALUE_SELECTOR_LOWER_AND_UPPER

最小値と最大値の平均値以下を試し、次にその値より大きい値を試す。

CS_VALUE_SELECTOR_UPPER_AND_LOWER

最小値と最大値の平均値以上を試し、次にその値より小さい値を試す。

CS_VALUE_SELECTOR_MEDIAN_AND_REST

変数の領域の中央の値を試し、次に中央の値を取り除く。

7.6.2 ユーザ定義の領域縮小方法

組み込みの領域縮小方法外に、ユーザ独自の領域縮小方法指定を定義することもできます。このためには、*cs_createValueSelector()* で独自定義の関数を登録した CSvalueSelector 型の変数を生成し、*CSvalueSelection* 型の変数を解探索関数に返す必要があります。

CSvalueSelector* **cs_createValueSelector** (IZBOOL (**init*)(int index, CSint** vars, int size, void* pData),IZBOOL (*next)(*CSvalueSelection** r, int index, CSint** vars, int size, void* pData),IZBOOL (*end)(int index, CSint** vars, int size, void* pData))

独自の領域縮小方法を CSvalueSelector 型変数として定義します。*init*, *next*, *end* はそれぞれ初期化、領域縮小方法の取得、終了処理を行う関数をそれぞれ指定します。これらの関数の引数および返り値は以下の通りです。

- init (一つの変数の領域縮小を開始する前に一回だけ呼ばれる関数)
  - *index* : 領域縮小対象となる変数の配列 vars 内の添え字
  - *vars* : 探索対象となる変数の配列
  - *size* : *vars* のサイズ
  - *pData* : ポインタまたは整数を格納できる領域へのポインタ。状態を保持するための変数として利用できる。
  - 返り値: 成功ならば TRUE

- next (一つの変数の領域縮小を順次試していくために呼ばれる関数)
  - *r* : 領域の縮小方法を受け取る *CSvalueSelection* 型変数へのポインタ。
  - *index* : 領域縮小対象となる変数の配列 vars 内の添え字
  - *vars* : 探索対象となる変数の配列
  - *size* : *vars* のサイズ
  - *pData* : ポインタまたは整数を格納できる領域へのポインタ。状態を保持するための変数として利用できる。
  - 返り値: 有効な r を返した場合は TRUE
- end (一つの変数の領域縮小を終了する前に一回だけ呼ばれる関数)
  - *index* : 領域縮小対象となる変数の配列 vars 内の添え字
  - *vars* : 探索対象となる変数の配列
  - *size* : *vars* のサイズ
  - *pData* : ポインタまたは整数を格納できる領域へのポインタ。状態を保持するための変数として利用できる。メモリを malloc などで割り当てていた場合は解放する必要がある。
  - 返り値: 成功ならば TRUE

返された CSvalueSelector 型変数は、不要になったら *cs_freeValueSelector()* で解放する必要があります。

void **cs_freeValueSelector** (CSvalueSelector* *vs*)

cs_createValueSelector() で生成された CSvalueSelector 型の変数を解放します。

7.6.3 領域縮小方法に関連する型および関数

CSvalueSelection

CSvalueSelection 型は変数の領域を縮小する方法を表現するもので、以下のように定義されています。:

```
typedef struct {
  int method;
  int value;
} CSvalueSelection;
```

value フィールドに指定された値を使用して method フィールドで指定された方法で変数の領域を縮小することを表現します。

method フィールドは以下の値をとります。

CS_VALUE_SELECTION_EQ
value に即値化することで領域を縮小する。

CS_VALUE_SELECTION_NEQ
value を取り除くことで領域を縮小する。

CS_VALUE_SELECTION_LE
value 以下とすることで領域を縮小する。

CS_VALUE_SELECTION_LT
value より小さくすることで領域を縮小する。

CS_VALUE_SELECTION_GE
value 以上とすることで領域を縮小する。

CS_VALUE_SELECTION_GT
value より大きくすることで領域を縮小する。

IZBOOL **cs_initValueSelector**(const CSvalueSelector* *vs*, int *index*, CSint**vars, int *size*, void* *pData*)
size 個の要素を持つ配列 *vars* の添え字 *index* で指定された CSint 型変数に対して、領域の縮小を行うための準備を行います。*pData* は、ポインタと int を格納するために十分な大きさを持ったメモリ領域へのポインタです。

成功した場合は TRUE が返されます。

IZBOOL **cs_selectNextValue**(*CSvalueSelection** *r*, const CSvalueSelector* *vs*, int *index*, CSint** *vars*, int *size*, void* *pData*)
size 個の要素を持つ配列 *vars* の添え字 *index* で指定された CSint 型変数に対して、領域の縮小方法を :c:type:CSvalueSelection: 型変数 *r* に返します。*pData* には *cs_initValueSelector()* で渡したポインタを指定します。

この関数は呼び出される度に次の領域の縮小方法を *r* に返そうとし、返せた場合は戻り値が TRUE です。全ての領域の縮小方法を返し終わっていたら FALSE を返します。

IZBOOL **cs_endValueSelector**(const CSvalueSelector* *vs*, int *index*, CSint** *vars*, int *size*, void* *pData*)
cs_initValueSelector() で開始された領域の縮小方法の列挙を終了します。*pData* には *cs_initValueSelector()* で渡したポインタを指定します。

成功した場合は TRUE が返されます。

IZBOOL **cs_selectValue**(CSint* *v*, const *CSvalueSelection** *vs*)
vs で指定された領域の縮小操作を CSint 型変数 *v* に適用します。成功した場合は TRUE が返されます。

7.7 NoGood の管理

cs_searchValueSelectorRestartNG() ではフェイルした変数と値の組合せを NoGood として CSnoGoodSet 型の変数に保存します。

CSnoGoodSet* **cs_createNoGoodSet** (CSint** *vars*, int *size*, IZBOOL (**prefilter*)(CSnoGoodSet* ngs, CSnoGood* ng, CSint** vars, int size, void* ext), int maxNoGood,void(*destructorNotify)(CSnoGoodSet* ngs, void* ext), void* *ext*)

フェイルが発生した変数と値の組合せを記録するための領域を作成します。

引数の意味はそれぞれ以下の通りです。

- *vars* : 解探索関数に与える CSint 型変数の配列です。
- *size* : 解探索関数に与える CSint 型変数の配列のサイズです。
- *prefilter* : NoGood を登録するかどうかを判定する関数を指定します。関数が TRUE を返した場合は NoGood が登録されます。*prefilter* に NULL を指定した場合は常に登録されます。
- *maxNoGood* : 登録できる NoGood の最大数です。最大数に到達した場合、重要でない NoGood から削除されます。
- *destructorNotify* : *cs_createNoGoodSet()* で作成された領域は *cs_restoreContext()* などで現在のコンテキストから戻る際に自動的に解放されますが、その時に *destructorNotify* で指定された関数が呼び出されます。関数呼び出しが不要な場合は NULL を指定します。
- *ext* : *prefilter* および *destructorNotify* の引数として使用されます。

prefilter 呼び出し時の引数はそれぞれ以下の通りです。

- *ngs* : この *cs_createNoGoodSet()* 呼び出しで作成された領域です。
- *ng* : 登録しようとしている NoGood へのポインタです。
- *vars*, *size* および *ext* : この *cs_createNoGoodSet()* 呼び出し時に使用されたものです。

destructorNotify 呼び出し時の引数はそれぞれ以下の通りです。

- *ngs* : この *cs_createNoGoodSet()* 呼び出しで作成された領域です。
- *vars*, *size* および *ext* : この *cs_createNoGoodSet()* 呼び出し時に使用されたものです。

int **cs_getNbNoGoods** (CSnoGoodSet* *ngs*)

ngs に登録されている NoGood の数を返します。

int **cs_getNbNoGoodElements** (CSnoGood* *ng*)

ng に登録されている要素の数を返します。一つの要素は以下から構成され、*cs_getNoGoodElementAt()* で取得することが可能です。

void **cs_getNoGoodElementAt** (int* *pIndex*, *CSvalueSelection** *vs*, CSnoGood* *ng*, int *index*)

ng に登録されている *index* 番目の要素を確認します。

- *pIndex* に `cs_createNoGoodSet()` の引数 *vars* 内での位置が格納されます。
- *vs* に領域縮小方法 (何をした時にフェイルしたか) が格納されます。

void **cs_filterNoGood** (CSnoGoodSet* *ngs*, IZBOOL (**filter*)(CSnoGoodSet* ngs, CSnoGood* elem, CSint** vars, int size, void* ext), void* *ext*)

ngs に登録されている NoGood 全体をスキャンし、*filter* 関数が FALSE を返した NoGood を削除します。*filter* に渡される引数は `cs_createNoGoodSet()` の *prefilter* に渡されるものと同じです。

void **cs_addNoGood** (CSnoGoodSet* *ngs*, int* *index*, *CSvalueSelection** *vs*, int *size*)

ngs に NoGood を追加します。*index* および *vs* はそれぞれ変数を指定する添え字と変数の領域縮小方法を指定する配列で、要素は *size* 個です。

void **cs_setMaxNbNoGoods** (CSnoGoodSet* *ngs*, int *max*)

ngs に保存できる NoGood の個数を *max* 個に変更します。

7.8 探索状態の通知

CSsearchNotify 型の変数を引数としてとる解探索関数は、コールバック関数の呼び出しを通じて探索中の状態を通知することができます。

CSsearchNotify* **cs_createSearchNotify** (void* *ext*)

解探索関数に渡すための CSsearchNotify 型変数を生成します。引数 *ext* はコールバック関数の呼び出し時に渡されます。

返された CSsearchNotify 型変数は、不要になったら `cs_freeSearchNotify()` で解放する必要があります。

void **cs_freeSearchNotify** (CSsearchNotify* *notify*)

`cs_createSearchNotify()` で生成された CSsearchNotify 型変数を解放します。

void **cs_searchNotifySetSearchStart** (CSsearchNotify* *notify*, void (**searchStart*)(int maxFails, CSint** allvars, int nbVars, void* ext))

解探索の開始時に呼び出される関数 *searchStart* を登録します。

`cs_searchValueSelectorRestartNG()` では、リスタートの単位ごとに呼び出されます。

searchStart には以下が引数として渡されます。

- *maxFails* : これから行う探索に許されるフェイル数の上限
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ

- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetSearchEnd**(CSsearchNotify* *notify*, void (**searchEnd*)(IZBOOL result, int nbFails, int maxFails, CSint** allvars, int nbVars, void* ext))

解探索の終了時に呼び出される関数 *searchEnd* を登録します。

`cs_searchValueSelectorRestartNG()` では、リスタートの単位ごとに呼び出されます。

searchEnd には以下が引数として渡されます。

- *result* : 解が見つかったら TRUE さもなければ FALSE
- *nbFails* : 探索中に起こったフェイルの数
- *maxFails* : 探索に許されていたフェイル数の上限
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetBeforeValueSelection**(CSsearchNotify* *notify*, void (**beforeValueSelection*)(int depth, int index, const *CSvalueSelection** vs, CSint** allvars, int nbVars, void* ext))

領域縮小を行う直前に呼び出される関数 *beforeValueSelection* を登録します。

beforeValueSelection には以下が引数として渡されます。

- *depth* : 探索の深さ
- *index* : 選択した変数の添え字
- *vs* : 領域縮小方法
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetAfterValueSelection**(CSsearchNotify* *notify*, void (**afterValueSelection*)(IZBOOL result, int depth, int index, const *CSvalueSelection** vs, CSint** allvars, int nbVars, void* ext))

領域縮小を行なった結果を通知するための関数 *afterValueSelection* を登録します。

afterValueSelection には以下が引数として渡されます。

- *result* : 領域縮小ができたら TRUE、フェイルしたら FALSE

- *depth* : 探索の深さ
- *index* : 選択した変数の添え字
- *vs* : 領域縮小方法
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetEnter**(CSsearchNotify* *notify*, void (**enter*)(int depth, int index, CSint** allvars, int nbVars, void* ext))

領域縮小を行う変数が決定した時に呼び出される関数 *enter* を登録します。

enter には以下が引数として渡されます。

- *depth* : 探索の深さ
- *index* : 選択した変数の添え字
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetLeave**(CSsearchNotify* *notify*, void (**leave*)(int depth, int index, CSint** allvars, int nbVars, void* ext))

一つの変数について、全ての領域縮小を試した後に呼び出される関数 *leave* を登録します。

leave には以下が引数として渡されます。

- *depth* : 探索の深さ
- *index* : 選択した変数の添え字
- *allvars* : 解探索関数に渡された領域変数の配列
- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

void **cs_searchNotifySetFound**(CSsearchNotify* *notify*, IZBOOL (**found*)(int depth, CSint** allvars, int nbVars, void* ext))

全ての変数を即値化できた時に呼び出される関数 *found* を登録します。

found には以下が引数として渡されます。

- *depth* : 探索の深さ
- *allvars* : 解探索関数に渡された領域変数の配列

- *nbVars* : 解探索関数に渡された領域変数の配列のサイズ
- *ext* : `cs_createSearchNotify()` に渡されたポインタ

found が TRUE を返すと、解探索関数は TRUE を返して終了します。FALSE を返した場合、探索を続行します。

リスタートを行う解探索関数では、同じ解について *found* が複数回呼び出される可能性があることに注意してください。

第 8 章

デモン

制約伝播の際には CSint 型変数の領域の変化を表わす幾つかのイベントが発生します。イベントは排他的な以下の 4 種類に分類されます。

Known: 領域は 1 つの値に縮小された (CSint 型変数が即値化された場合です。)

Ex.: `cs_EQ(vint, 0); /* Known` **イベントが発生する例** `*/`

NewMin: 領域の下限が変更された (必ず以前より大きな値に変化します。)

Ex.: `cs_GT(vint, 0);`

NewMax: 領域の上限が変更された (必ず以前より小さな値に変化します。)

Ex: `cs_LT(vint, 0);`

Neq: 領域から 1 つの値が取り除かれた

Ex: `cs_NEQ(vint, 0);`

これらのイベントが排他的であるとは、以下のことを意味します。

- known イベント発生する場合には、newMin, newMax, neq イベントは発生しません。
- newMin イベントおよび newMax イベントが発生する場合には、neq イベントは発生しません。

IZBOOL **`cs_eventAllKnown`** (CSint *******array*, int *size*, IZBOOL (**allKnown*)(CSint **array, int size, void *extra), void **extra*)

allKnown() 関数は *array* によって参照されるすべての CSint 型変数が即値化した (つまり known イベントが発生した) 時に呼び出されます。

cs_eventAllknown() が呼び出されたときにすでに *array* で参照できる CSint 型変数がすべて即値化されている場合には、直ちに allKnown() が呼ばれ、その結果が返ります。 そうでなければ、本関数は単に TRUE を返します。

引数 *extra* は自由に使うことのできる void 型変数へのポインタです。allKnown(array, size, extra) が呼ばれる時に、引数 *extra* として cs_eventAllKnown(array, size, allKnown, extra) で渡された *extra* が渡されます。

IZBOOL **cs_eventKnown**(CSint **_array_, int _size_, IZBOOL (*_known_)(int val, int index, CSint **array, int size, void *extra), void *_extra_)

CSint 型変数 _array_ [_index_] が即値化された時に、関数 known() が呼び出されます。

引数 _extra_ は自由に使うことのできる void 型変数へのポインタです。known(val, index, array, size, extra) が呼ばれる時に、引数 _extra_ として cs_eventKnown(array, size, known, extra) で渡された _extra_ が渡されます。

void **cs_eventNewMin**(CSint **_array_, int _size_, IZBOOL (*_newMin_)(CSint *vint, int index, int oldMin, CSint **array, int size, void *extra), void *_extra_)

CSint 型変数 _array_ [_index_] の領域の下限が増大するたびに関数 _newMin()_ が呼び出されます。

引数 _extra_ は自由に使うことのできる void 型変数へのポインタです。newMin(vint, index, oldMin, array, size, extra) が呼ばれる時に、引数 _extra_ として cs_eventNewMin(array, size, newMin, extra) で渡された _extra_ が渡されます。

void **cs_eventNewMax**(CSint **_array_, int _size_, IZBOOL (*_newMax_)(CSint *vint, int index, int oldMax, CSint **array, int size, void *extra), void *_extra_)

CSint 型変数 _array_ [_index_] の領域の上限が減少するたびに関数 _newMax()_ が呼び出されます。

引数 _extra_ は自由に使うことのできる void 型変数へのポインタです。newMax(vint, index, oldMax, array, size, extra) が呼ばれる時に、引数 _extra_ として cs_eventNewMax(array, size, newMax, extra) で渡された _extra_ が渡されます。

void **cs_eventNeq**(CSint **_array_, int _size_, IZBOOL (*_neq_)(CSint *vint, int index, int neqValue, CSint **array, int size, void *extra), void *_extra_)

CSint 型変数 _array_ [_index_] の領域の上限・下限以外の値が取り除かれるたびに関数 _neq()_ が呼び出されます。(ただし即値化された時を除きます。)

引数 _extra_ は自由に使うことのできる void 型変数へのポインタです。neq(vint, index, neqValue, array, size, extra) が呼ばれる時に、引数 _extra_ として cs_eventNeq(array, size, neq, extra) で渡された _extra_ が渡されます。

また探索中の状態に関連したイベントの発生を知るための関数が存在します。

void **cs_backtrack**(CSint *_vint_, int _i_, void (*_backtrack_)(CSint *vint, int i))

バックトラックが発生し本関数の呼び出しコンテキストに戻った時に、引数で渡された CSint 型変数および整数を引数として、関数 _backtrack()_ が呼び出されます。

void **cs_backtrackExt**(CSint *_vint_, void* _ext_, void (*_backtrack_)(CSint *vint, void *ext))

バックトラックが発生し本関数の呼び出しコンテキストに戻った時に、引数で渡された CSint 型変数およびポインタを引数として、関数 _backtrack()_ が呼び出されます。

void **cs_eventConflict**(CSint **_array_, int _size_, void (*_conflict_)(CSint *vint, int index, const _CSvalueSelection_* vs, CSint **array, int size, void *extra), void *_extra_)

CSint 型変数 _array_ 内の変数で制約伝播が失敗したときに関数 _conflict()_ が呼び出されます。引数 _extra_ は自由に使うことのできる void 型変数へのポインタです。

conflict() が呼び出されたとき、引数は以下のようになります。

- _vint_, _index_ :制約伝播を引き起こした変数へのポインタと array での位置

- *vs* : 変数に対して行った領域縮小の操作

その他の引数は、cs_eventConflict 呼び出し時に使用されたものと同じです。

第 9 章

コンテキスト

以下の関数はすべての CSint 型変数に発生したイベントについての情報を保持しているヒープ領域にアクセスするためのものです。ここでいうイベントとは、制約伝播により引き起こされる CSint 型変数の領域の変化 (known, newMin. newMax, neq) であるか、制約の設定を指します。.

cs_search() のような木探索のための組み込み関数を使用する場合には、以下の関数を利用することはありませんが、探索のための関数を自分で作成するような場合には、以下の関数を利用してバックトラックの制御を行うことができます。

int **cs_saveContext**()

コンテキストを保存します。これにより、保存した時点のすべての CSint 型変数の状態および制約の状態に戻ることができるようになります。

この関数が返す整数は *label* として、*cs_forgetSaveContextUntil()*, *cs_acceptContextUntil()* あるいは *cs_restoreContextUntil()* を呼び出す時に引数として渡す必要があります。

void **cs_forgetSaveContext**()

直前の *cs_saveContext()* の呼び出しをキャンセルします。 cs_event() 関数中では、*cs_acceptContext()* ではなく、本関数を使わなくてはなりません。

void **cs_restoreAndSaveContext**()

コンテキストを直前の *cs_saveContext()* 呼び出し以前と同じ状態に戻し、再び *cs_saveContext()* を呼び出します。

void **cs_acceptContext**()

直前の *cs_saveContext()* 呼び出し以降に CSint 型変数に発生したのすべての変化を受け入れます (コンテキストはもう元には戻せなくなります)。

この関数は直前の *cs_saveContext()* 呼び出し以降にヒープ領域に確保されたメモリを解放します。

直前の *cs_saveContext()* 呼び出し時点へのバックトラックができなくなってしまいますので *cs_acceptContext()* は cs_event() 関数中で呼び出していけません。

void **cs_acceptAll**()
: 最初の *cs_saveContext()* 呼び出し以降に CSint 型変数に発生したのすべての変化を受け入れます。この関数はヒープ領域に確保されたすべてのメモリを解放します。

void **cs_restoreContext**()
: コンテキストを直前の *cs_saveContext()* 呼び出し以前と同じ状態に戻します。

void **cs_restoreAll**()
: コンテキストを最初の *cs_saveContext()* 呼び出し以前と同じ状態に戻します。

以下の 3 つの関数は引数として int 型のラベルをとります。このラベルは事前に呼び出された *cs_saveContext()* が返したものでなくてはなりません。

void **cs_forgetSaveContextUntil**(int *label*)
: *label* によって参照される状態までのすべての *cs_saveContext()* 呼び出しを無効にします。

void **cs_acceptContextUntil**(int *label*)
: *label* によって参照される *cs_saveContext()* 呼び出し以降に CSint 型変数に発生したのすべての変化を受け入れます。(受け入れたコンテキストはもう元に戻せなくなります。)

  この関数は label によって参照される *cs_saveContext()* 呼び出し以降にヒープ領域に確保されたメモリを解放します。

void **cs_restoreContextUntil**(int *label*)
: コンテキストは *label* により参照される *cs_saveContext()* の呼び出しより前の状態に戻されます。

例：cs_search() の手続きは、以下のように実装することができます。:

```
IZBOOL cs_search(CSint **allvars, int nbVars, CSint* (*findFreeVar)(CSint **allvars,␣
↪int nbVars)) {
  CSint *var = findFreeVar(allvars, nbVars);
  if (var) {
    int val;
    cs_saveContext();

    for (val = cs_getMin(var); val <= cs_getMax(var); val = cs_getNextValue(var, val))
↪{
      if (cs_EQ(var, val) && cs_search(allvars, nbVars, findFreeVar))
        return TRUE;

      cs_restoreAndSaveContext();
    }

    cs_acceptContext();
    return FALSE;
  }
  else {
    return TRUE;
```

(次のページに続く)

(前のページからの続き)

```
    }
}
```

索引