
iZ-C Reference Manual

version 3.5

NTT DATA SEKISUI SYSTEMS

2017 年 12 月 19 日

Contents:

第 1 章	はじめに	1
第 2 章	実行環境	3
2.1	開始と終了	3
2.2	エラーの取得	3
2.3	エラー発生時の挙動の設定	4
2.4	バージョン情報の取得	4
2.5	統計情報の出力	5
第 3 章	コンストラクタと基本的な関数	7
3.1	コンストラクタ	7
3.2	領域変数の情報にアクセスするための関数	7
3.3	領域に関する制約	9
第 4 章	算術制約	11
第 5 章	関係制約	13
第 6 章	高水準制約	17
第 7 章	解探索とヒューリスティクス	19
7.1	基本的な解探索	19
7.2	組み込みの選択関数	19
7.3	フェイル数とチョイスポイント数	20
7.4	複雑な解探索関数	21
第 8 章	デモン	25
第 9 章	コンテキスト	27
	索引	31

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()

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()

`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()

`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()

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_setErrorHandler** (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_setErrorHandler` 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** ()

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** ()

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

コンストラクタと基本的な関数

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い場合は TRUE を返します。そうでなければ FALSE を返します。

「即値化されていない」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。

「即値化されている」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。

$$\text{cs_VAdd}(n, V1, V2, \dots, Vn) = V1 + V2 + \dots + Vn$$

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、 $V1 - V2 - V3 - \dots - Vn$ であるような CSint 型変数を返します。

$$\text{cs_VSub}(n, V1, V2, V3, \dots, Vn) = V1 - V2 - V3 - \dots - Vn$$

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 (CSint 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* より大きいかに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ものではありません。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。

$$i \in \text{domain}(\text{Index}) \Leftrightarrow \text{val} \in \text{domain}(\text{array} [i])$$

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_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)。

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。
2. 見つけた変数 *var* の領域から値 *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)`

即値化に失敗した場合は `cs_NEQ(var, val)` としてステップ 2 に戻ります。

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、最初に見つけた (=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_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_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 NbFails-
Max)

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", 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", 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;  
}
```


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。

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、backtrack() 関数が呼び出されます。

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うので *cs_acceptContext()* は cs_event() 関数中で呼び出してはいけません。

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;
    }
}
```



```
}  
}
```


索引

cs_Abs (C の関数), 12
 cs_acceptAll (C の関数), 27
 cs_acceptContext (C の関数), 27
 cs_acceptContextUntil (C の関数), 28
 cs_Add (C の関数), 11
 cs_AllNeq (C の関数), 14
 cs_backtrack (C の関数), 26
 cs_createCSint (C の関数), 7
 cs_createCSintArray (C の関数), 7
 cs_createCSintFromDomain (C の関数), 7
 cs_createNamedCSint (C の関数), 7
 cs_Div (C の関数), 12
 cs_Element (C の関数), 18
 cs_end (C の関数), 3
 cs_Eq (C の関数), 13
 CS_ERR_GETVALUE (C のマクロ), 4
 CS_ERR_NO_MEMORY (C のマクロ), 4
 CS_ERR_NONE (C のマクロ), 4
 CS_ERR_OVERFLOW (C のマクロ), 4
 cs_eventAllKnown (C の関数), 25
 cs_eventKnown (C の関数), 26
 cs_eventNeq (C の関数), 26
 cs_eventNewMax (C の関数), 26
 cs_eventNewMin (C の関数), 26
 cs_findAll (C の関数), 22
 cs_findFreeVar (C の関数), 19
 cs_findFreeVarNbConstraints (C の関数), 20
 cs_findFreeVarNbElements (C の関数), 19
 cs_findFreeVarNbElementsMin (C の関数), 20
 cs_forgetSaveContext (C の関数), 27
 cs_forgetSaveContextUntil (C の関数), 28
 cs_fprintf (C の関数), 9
 cs_fprintStats (C の関数), 5
 cs_freeDomain (C の関数), 8
 cs_Ge (C の関数), 13
 cs_getDomain (C の関数), 8
 cs_getErr (C の関数), 3
 cs_getMax (C の関数), 8
 cs_getMin (C の関数), 7
 cs_getName (C の関数), 8
 cs_getNbChoicePoints (C の関数), 21
 cs_getNbConstraints (C の関数), 8
 cs_getNbElements (C の関数), 8
 cs_getNbFails (C の関数), 20
 cs_getNextValue (C の関数), 8
 cs_getPreviousValue (C の関数), 8
 cs_getValue (C の関数), 9
 cs_getVersion (C の関数), 4
 cs_Gt (C の関数), 13
 cs_IfEq (C の関数), 17
 cs_IfNeq (C の関数), 17
 cs_InArray (C の関数), 9
 cs_Index (C の関数), 17
 cs_InInterval (C の関数), 10
 cs_init (C の関数), 3
 cs_initErr (C の関数), 3
 cs_isFree (C の関数), 9

cs_isIn (C の関数), 8
 cs_isInstantiated (C の関数), 9
 cs_Le (C の関数), 13
 cs_Lt (C の関数), 13
 cs_Max (C の関数), 18
 cs_Min (C の関数), 18
 cs_minimize (C の関数), 22
 cs_Mul (C の関数), 11
 cs_Neq (C の関数), 13
 cs_NotInArray (C の関数), 9
 cs_NotInInterval (C の関数), 10
 cs_Occur (C の関数), 17
 cs_OccurConstraints (C の関数), 17
 cs_OccurDomain (C の関数), 17
 cs_printf (C の関数), 9
 cs_printStats (C の関数), 5
 cs_ReifEq (C の関数), 14
 cs_ReifGe (C の関数), 14
 cs_ReifGt (C の関数), 14
 cs_ReifLe (C の関数), 14
 cs_ReifLt (C の関数), 14
 cs_ReifNeq (C の関数), 14
 cs_restoreAll (C の関数), 28
 cs_restoreAndSaveContext (C の関数), 27
 cs_restoreContext (C の関数), 28
 cs_restoreContextUntil (C の関数), 28
 cs_saveContext (C の関数), 27
 cs_ScalProd (C の関数), 12
 cs_search (C の関数), 19
 cs_searchCriteria (C の関数), 21
 cs_searchCriteriaFail (C の関数), 21
 cs_searchFail (C の関数), 21
 cs_searchMatrix (C の関数), 21
 cs_searchMatrixFail (C の関数), 21
 cs_setErr (C の関数), 4
 cs_setErrorHandler (C の関数), 4
 cs_setName (C の関数), 9
 cs_Sigma (C の関数), 12
 cs_Sub (C の関数), 11
 cs_VAdd (C の関数), 11
 cs_VarElement (C の関数), 18
 cs_VDiv (C の関数), 12
 cs_VMax (C の関数), 18
 cs_VMin (C の関数), 18
 cs_VMul (C の関数), 11
 cs_VScalProd (C の関数), 12
 cs_Vsearch (C の関数), 19
 cs_VSub (C の関数), 11
 CSINT (C の関数), 7

IZ_VERSION_MAJOR (C のマクロ), 4
 IZ_VERSION_MINOR (C のマクロ), 4
 IZ_VERSION_PATCH (C のマクロ), 5