Nice Loop 入門

Nice Loop は、ふたつのセル（マス）間の論理的依存関係の連鎖を利用してセルの候補数字を特定したり削除する手筋です。いわゆる chain 系手筋のひとつで X-cycle や XY-chain をさらに一般化した手筋です。n 国同盟や n-grid、Pattern Overlay Method などでも盤面を進められない状況でも候補数字を特定したり絞り込める場合があります。

NumberPlace.lisp では、この Nice Loop を連鎖セル数無制限という条件で実装しました。連鎖セル数を任意の長さに制限することも可能です（3 未満はループを構成できないので無意味）。上級者向けの設定である「advanced-level」では連鎖セル数上限=3、超上級者向け設定である「machine-level」では連鎖セル数上限=無制限と設定しています（以下では、青字部分がユーザが入力する部分を表しています）。

cl-user> (advanced-level)　---連鎖セル数上限を 3 に制限した状態で Nice Loop を使用。

cl-user> (machine-level)　---連鎖セル数上限を無制限にした状態で Nice Loop を使用。

Nice Loop を使用できるのは上記の「advanced-level」と「machine-level」に限定してありますが、「senior-level」以下の設定でも使用を許可する手筋に Nice Loop を簡単に追加可能です。また連鎖セル数の制限も自由に設定できます。たとえば NumberPlace.lisp のソースコード中で関数「senior-level」は以下のように定義されています（コメント部分は省略）。

```
(defun senior-level (&optional (depth nil))
  (easy-method-first nil)
  (auto-trim-level 100)
  (chain-trim t)
  (trim-every-time t)
  (n-grid-limit 2)
  (tuples-limit 3)
  (think-depth depth)
  (permitted-methods
   '(do-primary do-only-one do-cell-unique do-localization do-n-
tuples do-n-grid))
  (print-preset-level))
```

Nice Loop を許可して連鎖セル数上限を 8 に設定する場合は以下のようにします。

```
(defun senior-level (&optional (depth nil))
  (easy-method-first nil)
  (auto-trim-level 100)
  (chain-trim t)
  (trim-every-time t)
  (n-grid-limit 2)
```

```
  (tuples-limit 3)
  (max-nice-length 8)
  (capital-address t)              ;セル・アドレスを大文字で表示する。
  (print-with-symbol-letter t) ;Nice Loop 経路を表示する際にラベル記号も表示する。
  (think-depth depth)
  (permitted-methods
   '(do-primary do-only-one do-cell-unique do-nice-loop do-localization do-n-tuples do-n-grid))
  (print-preset-level))
```

青字の 2 行目と 3 行目はオプション設定なのでなくても構いません。許可する手筋の一覧は順序も意味を持ちます。Nice Loop では 2 値セルと 2 所セル（後述の「定義」を参照）を手掛かりにループを構成するので例で指定したあたりの順序が良いでしょう。

数独（ナンプレ）問題の与え方

Nice Loop とは無関係ですが基本なので念のために紹介します。問題の与え方は大きく分けて 2 種類あります。ひとつは画面上から直接入力する方法で、もうひとつは、あらかじめファイルに書式にしたがって問題を登録しておいて読み込みコマンドでまとめて読み込む方法です。どちらの方法にも共通する 2 種類の書式があります。

ひとつめは Common Lisp の 2 次元配列として数独（ナンプレ）の問題を与える方法です。カッコを多用するので慣れないと煩雑ですが、数種類に絞り込んだ候補数字をセルごとに指定することも出来る汎用的な書式です。

もうひとつは盤面上の未確定値を「0」として盤面全体の確定値を 81 桁の整数として与える方法です。NumberPlace.lisp では、このような書式のデータを「chunk（かたまり）」と呼んでいます。「board 型データでの入力方法」で紹介している数独（ナンプレ）の問題の場合、chunk 形式では、これを「100007090030020008009600500005300900010080002600004000300000010040000010007000300」と書きます。インターネット上で使われていることも多く、候補数字を表現できないという問題があるものの、コピー&ペーストで簡単に扱えるので便利です。

NumberPlace.lisp では Common Lisp の 2 次元配列としての汎用的書式（board 型と呼んでいます）と chunk 形式データを相互に変換する関数を用意しています。以下の問題を例にそれぞれの形式での入力方法を紹介します。

board 型データでの入力方法

```
+-------+-------+-------+
| 1 - - | - - 7 | - 9 - |
| - 3 - | - 2 - | - - 8 |
| - - 9 | 6 - - | 5 - - |
```

```
+-------+-------+-------+
| - - 5 | 3 - - | 9 - - |
| - 1 - | - 8 - | - - 2 |
| 6 - - | - - 4 | - - - |
+-------+-------+-------+
| 3 - - | - - - | - 1 - |
| - 4 - | - - - | - - 7 |
| - - 7 | - - - | 3 - - |
+-------+-------+-------+
```

という問題を画面上で直接入力する場合は入力した問題を保存する変数（たとえば p-01）を指定して「enter-board」関数を以下のように使用します（青字部分が入力する部分。**太字**は NumberPlace.lisp からのプロンプト）。

```
cl-user> (setf p-01 (enter-board))
```

表示されるプロンプトにしたがってデータを入力します。

```
ボードのサイズを指定してください。
9
行ごとに入力します。空欄には「0」を入力してください。
1 行目: 1 0 0 0 0 7 0 9 0
2 行目: 0 3 0 0 2 0 0 0 8
3 行目: 0 0 9 6 0 0 5 0 0
4 行目: 0 0 5 3 0 0 9 0 0
5 行目: 0 1 0 0 8 0 0 0 2
6 行目: 6 0 0 0 0 4 0 0 0
7 行目: 3 0 0 0 0 0 0 1 0
8 行目: 0 4 0 0 0 0 0 1 0
9 行目: 0 0 7 0 0 0 3 0 0

#2A((1 0 0 0 0 7 0 9 0)
    (0 3 0 0 2 0 0 0 8)
    (0 0 9 6 0 0 5 0 0)
    (0 0 5 3 0 0 9 0 0)
    (0 1 0 0 8 0 0 0 2)
    (6 0 0 0 0 4 0 0 0)
    (3 0 0 0 0 0 0 1 0)
    (0 4 0 0 0 0 0 1 0)
    (0 0 7 0 0 0 3 0 0))
cl-user>
```

変数「p-01」にデータが保存されていることを確認してみます。

```
cl-user> p-01
#2A((1 0 0 0 0 7 0 9 0)
    (0 3 0 0 2 0 0 0 8)
    (0 0 9 6 0 0 5 0 0)
    (0 0 5 3 0 0 9 0 0)
    (0 1 0 0 8 0 0 0 2)
    (6 0 0 0 0 4 0 0 0)
    (3 0 0 0 0 0 0 1 0)
    (0 4 0 0 0 0 0 0 7)
    (0 0 7 0 0 0 3 0 0))
cl-user>
```

ここで表示されているのが Common Lisp での 2 次元配列のデータ表現そのものです。もし特定のセルに候補数字を与えたい場合は候補数字群をカッコで囲んで与えます。例えば 1 行 2 列の候補数字に「5」と「7」を与えたいのであれば先ほどの 1 行目の入力を次のようにします。

```
1 行目: 1 (5 7) 0 0 0 7 0 9 0
```

「edit-board」という関数を使うと入力したデータを後から修正することが出来ます。

```
cl-user> (edit-board p-01)
#=======================================================================#
#       |       |       #       |       |       #       |       |       #
#   1   |       |       #       |       |   7   #       |   9   |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   3   |       #       |   2   |       #       |       |   8   #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |       |   9   #   6   |       |       #   5   |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================#=======================#=======================#
#       |       |       #       |       |       #       |       |       #
#       |       |   5   #   3   |       |       #   9   |       |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   1   |       #       |   8   |       #       |       |   2   #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#   6   |       |       #       |       |   4   #       |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================#=======================#=======================#
#       |       |       #       |       |       #       |       |       #
#   3   |       |       #       |       |       #       |   1   |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   4   |       #       |       |       #       |   1   |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
```

```
#       |       |       #       |       |       #       |       |       #
#       |       |   7   #       |       |       #   3   |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================================================================#
修正する行と列の番号を指定してください：1 2
修正する値を指定してください：(5 7)
#=======================================================================#
#       |       |       #       |       |       #       |       |       #
#   1   |   5   |       #       |       |   7   #       |   9   |       #
#       | 7     |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   3   |       #       |   2   |       #       |       |   8   #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |       |   9   #   6   |       |       #   5   |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================#=======================#=======================#
#       |       |       #       |       |       #       |       |       #
#       |       |   5   #   3   |       |       #   9   |       |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   1   |       #       |   8   |       #       |       |   2   #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#   6   |       |       #       |       |   4   #       |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================#=======================#=======================#
#       |       |       #       |       |       #       |       |       #
#   3   |       |       #       |       |       #   1   |       |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |   4   |       #       |       |       #   1   |       |       #
#       |       |       #       |       |       #       |       |       #
#-------+-------+-------#-------+-------+-------#-------+-------+-------#
#       |       |       #       |       |       #       |       |       #
#       |       |   7   #       |       |       #   3   |       |       #
#       |       |       #       |       |       #       |       |       #
#=======================================================================#
続けますか？ (y/n) Please answer with y or n : n

#2A((1 (5 7) 0 0 0 7 0 9 0)
    (0 3 0 0 2 0 0 0 8)
    (0 0 9 6 0 0 5 0 0)
    (0 0 5 3 0 0 9 0 0)
    (0 1 0 0 8 0 0 0 2)
    (6 0 0 0 0 4 0 0 0)
    (3 0 0 0 0 0 0 1 0)
    (0 4 0 0 0 0 0 1 0)
    (0 0 7 0 0 0 3 0 0))
cl-user>
```

ファイルにデータを保存する場合は、この汎用的書式で書込んでおきます。その際データを保存す

る変数名もセットで指定しておくと便利です（青字が保存する変数名を指定するために追加する部分）。

```
(setf p-01
#2A((1 (5 7) 0 0 0 7 0 9 0)
    (0 3 0 0 2 0 0 0 8)
    (0 0 9 6 0 0 5 0 0)
    (0 0 5 3 0 0 9 0 0)
    (0 1 0 0 8 0 0 0 2)
    (6 0 0 0 0 4 0 0 0)
    (3 0 0 0 0 0 0 1 0)
    (0 4 0 0 0 0 0 1 0)
    (0 0 7 0 0 0 3 0 0))
)
```

複数のデータを保存する場合は「p-01」の部分を重複しない適当な変数名に変更します。変数名を指定する際、Common Lisp では大文字と小文字の区別はありませんので重複しないようご注意下さい。

これらのデータを保存したファイル名が「c:¥Users¥daigo¥Documents¥sudoku¥sudoku-01.lisp」だとすると、データの読み込みは次のようにします。

```
cl-user> (load  “c:¥Users¥daigo¥Documents¥sudoku¥sudoku-01.lisp” )
```

これでファイルに保存されているすべてのデータが読み込まれます。Common Lisp に詳しくない場合、ファイルへの保存はこの汎用的書式で行うことをお勧めします。以下の chunk 型データで保存した場合でもファイルから一括して読み込むことは可能ですが若干のプログラムが必要になります。

chunk 型データでの入力方法

chunk 型データを入力する場合は NumberPlace.lisp が用意する汎用形式への変換関数を使って入力します。

```
cl-user> (setf p-01 (chunk2board 1000070900300200080096005000053009000100800026000040003000000100400000100070003 00))
#2A((1 0 0 0 0 7 0 9 0)
    (0 3 0 0 2 0 0 0 8)
    (0 0 9 6 0 0 5 0 0)
    (0 0 5 3 0 0 9 0 0)
```

```
        (0 1 0 0 8 0 0 0 2)
        (6 0 0 0 0 4 0 0 0)
        (3 0 0 0 0 0 0 1 0)
        (0 4 0 0 0 0 0 1 0)
        (0 0 7 0 0 0 3 0 0))
    cl-user>
```

board 型のデータを chunk 型データに変換したいときは次のように「board2chunk」関数を使います。

```
    cl-user> (board2chunk p-01)
    "100007090030020008009600500005300900010080002600004000300000010040000010007000300"
    cl-user>
```

変数「p-01」に保存した問題を NumberPlace.lisp に解かせる場合は次のようにします。

```
    cl-user> (teach p-01)
```

解法過程で使用される手筋とその難易度をプロット表示するには

```
    cl-user> (plot p-01)
```

とします。

「teach」関数では解法過程の解説を解説盤面と共に表示します。Nice Loop を発見すると以下のように表示します。

```
            :
            :
            :
    不連続 Nice Loop  [(A)R5C2]-4-[(B)R6C3]-7-[(C)R4C3]-4-
    [(D)R4C8]=4=[(E)R5C8]-4-[(A)R5C2]
    ==> R5C2<>4
    +-------+-------+-------+
    | - - - | - - - | - - - |
    | - - - | - - - | - - - |
    | - - - | - - - | - - - |
    +-------+-------+-------+
    | - - C | - - - | - D - |
    | - A - | - - - | - E - |
    | - - B | - - - | - - - |
    +-------+-------+-------+
```

```
| - - - | - - - | - - - |
| - - - | - - - | - - - |
| - - - | - - - | - - - |
+-------+-------+-------+
0:0> Nice Loop により[@]の位置から候補を削除できます。
+-------+-------+-------+
| + + 9 | + + = | 7 + + |
| 7 + 3 | + = + | 4 @ + |
| + + = | 4 8 7 | + + + |
+-------+-------+-------+
| + + = | 5 1 6 | + + @ |
| + @ 2 | = + + | 6 + @ |
| + 6 = | + + + | + 5 @ |
+-------+-------+-------+
| 2 3 8 | 7 9 1 | 5 6 4 |
| + + @ | 8 = 3 | = + + |
| = 7 1 | 6 2 = | 3 8 9 |
+-------+-------+-------+
```

A, B, C, ・・・は連鎖の各セルに NumberPlace.lisp が一時的に与えたラベルで、アルファベット順に各セルが連鎖していることを示しています。盤面を表す図中のラベルと対応しています。

連鎖内のカッコに囲まれたラベルの直後にはセルの具体的アドレスが示されています。たとえば [(B)R6C3] は「B」というラベルが「6 行 3 列」に存在していることを表しています。行は横方向のセルの集まりで、列は縦方向のセルの集まりです。左上のセルが 1 行 1 列、右下のセルが 9 行 9 列となります。

上記の例の場合は Nice Loop の種類が「不連続 Nice Loop(Discontinuous Nice Loop)」で、「A」から「E」の 5 つのセルの連鎖からなり、結果として 5 行 2 列に候補数字「4」が存在することはあり得ないこと（R5C2<>4）を示しています。つまり 5 行 2 列の候補数字から 4 を削除できます。

大抵の盤面では複数の Nice Loop が成立するので、各 Nice Loop の内容を表示した後、それらの各 Nice Loop で削除できる候補をまとめて表示しています。NumberPlace.lisp は実際に削除できる候補だけを表示します。

以上が NumberPlace.lisp を使って Nice Loop を利用する場合に最低限必要な知識です。以下では自分自身で Nice Loop を発見しようとする際に必要となる知識を紹介しています。理論的根拠に興味がない場合は読む必要はありません。

Nice Loop を理解する

まず用語の定義を行います。

定義:ユニット(unit)
行・列・ブロックのいずれか。

定義:Bi-value セル(2 値セル)
候補数字が 2 つだけに絞り込まれたセルのこと。

定義:Bi-location セル(2 所セル)
2つのセル内に存在する共通の候補数字がユニット内の他のセルに存在しないとき、この2つのセルを Bi-location セルと呼ぶ。

定義:リンク
2つのセル A と B が同じユニットに属し、共通の候補数字を持つとき A と B はリンクしていると呼ぶ。

定義:Strong inference(強い演繹関係)
2つのセル A と B の間に次の関係があるとき A と B の関係を strong inference と呼ぶ。
When **A** is false, then **B** is true.

When **B** is false, then **A** is true.

(can not both be false)

A と B が同じユニットに属す場合、A と B が同時に true になることはない。したがって、

(A=False)→(B=True)

(A=True)→(B=False)

(B=False)→(A=True)

(B=True)→(A=False)

定義:Weak inference(弱い演繹関係)
2つのセル A と B の間に次の関係があるとき A と B の関係を Weak inference と呼ぶ。
When **A** is true, then **B** is false.

When **B** is true, then **A** is false.

(can not both be true)

A と B が同じユニットに属す場合でも、A と B が同時に false であることはあり得る。

(A=True)→(B=False)

(A=False)→(B=True/False)

(B=True)→(A=False)

(B=False)→(A=True/False)

A と B が同じユニットに属す場合、strong inference であれば weak inference である。した

がって**strong inference は weak inference でもある。**

定義：Strong link(強いリンク)
Strong inference を満たすリンクのこと。

定義：Weak link(弱いリンク)
Weak inference を満たすリンクのこと。

定義：連続的 nice loop (continuous nice loop)
先頭セルを含む、ループを構成するすべてのセルに対するリンクが**Nice Loop 連鎖ルール**（後述）に従っているのであれば、その Nice Loop を 連続的 Nice Loop と呼ぶ。

定義：不連続 nice loop (discontinuous nice loop)
先頭セル以外のすべてのループ構成セルが Nice Loop 連鎖ルールを満たし、先頭セルが Nice Loop 連鎖ルールを満たさないものの**Nice Loop 例外条件**（後述）は満たしているループを 不連続 Nice Loop と呼ぶ。

strong inference と weak inference は両方向に成立する

定義の中で特に重要なのは strong inference と weak inference という考え方です。図 1（後述）を例に考えてみます。R1C1（1 行 1 列のセル）と R1C2（1 行 2 列のセル）は 2 値セル、かつ 5 と6のふたつの候補数字を介した 2 所セルです。

R1C1 が 5 でないならば R1C2 は5ですし、6に関しても同様です。したがって strong inference の定義を満たしています。もちろん、この2つのセルは同じ行（この場合は同じブロックでもある）に存在し、リンクの定義も満たしているので strong link です。

R1C1 → R1C2 は strong inference

R1C2 が5でないならば R1C1 は 5 なので、逆向きにも、この関係は成り立っています。

R1C2 → R1C1 は strong inference

strong inference では一方向に成立した関係が逆向きにも成立しましたが、weak inference でも同じことが言えます。

たとえば図 1 の 2 行目には 1 列目と 8 列目、そして 9 列目（R2C1、R2C8、R2C9）の 3 箇所に候補数字 9 が存在します。このとき（どの2つを選んでも良いのですが）たとえば R2C1 が9であれば R2C8 は9ではありませんから R2C1 と R2C8 は 9 を介する weak inference です（weak inference の定義を確認してみて下さい）。

R2C1 → R2C8 は weak inference

逆向きの場合も R2C8 が9であるならば R2C1 は9ではありませんから、やはり weak inference です。

R2C8 → R2C1 は weak inference

このように strong inference の場合でも weak inference の場合でも、**一方向に成り立つ inference は両方向に成り立ちます。**

なお strong inference であれば weak inference の定義も満たすので **strong inferece を weak inference と解釈することもできます**。この点も重要なので記憶しておいて下さい。

連続的 Nice Loop と不連続 Nice Loop

2つのセル間の関係が strong inference の場合でも weak inference の場合でも、開始セルでの仮定が正しい限り、次のセルでの結論も正しいことが保証されます。先ほどの図 1 の R1C1 と R1C2 の例で言えば R1C1 が 5 でないならば R1C2 が 5 であるという結論は正しいですし、逆に R1C2 が5でないならば R1C1 が5であるという結論も正しく成立しています。

A というセルと B というセルに上記のリンク関係が成立していて、なおかつセル B とセル C にも別のリンク関係が成立しているのであれば、A での仮定は B での結論となり、B での結論が C に対する B での仮定となり、全体として A の仮定に対する C での結論となります。

このようなリンク関係の連鎖は連鎖するセル数がいくつに増えてもやはり成り立ちます。

A → B → C → ・・・ → Z

このような連鎖の場合、A での仮定が結局 Z での結論に直結します。この連鎖関係は双方向に成り立つので、Z での仮定は A での結論に直結しています。

Nice Loop の連鎖は strong link と weak link の組合せです。そして連鎖の開始セルと終端セルが一致するような連鎖を選ぶので全体としてループを構成します。

ループを構成しているので、もし終端セル（＝開始セル）で開始セルでの仮定に矛盾する結論が導かれたとすると開始セルでの仮定が誤りだったと言うことになります。たとえば連鎖の開始セルで「4 である」と仮定して連鎖を辿ると終端セル（=開始セル）で「4 でない」という結論が導かれたとすると、最初の「4である」という仮定に誤りがあったということです（数学的な意味での背理法）。つまり開始セルに4が含まれることはあり得ないので4を削除できます。

ループの開始セルでも「Nice Loop 連鎖ルール」を満たす場合を連続的 Nice Loop と呼びますが、この場合すべてのセル間のリンク関係の仮定は両方向に正しいことを意味しています。したがって、例えば2つのセル間のリンクが weak inference であるならば、2つのセル間の inference を介する候補数字（以下、ラベルと呼びます）は、その2つのセルのどちらかに存在しなければなりません。したがって、リンク・ラベルの表す候補数字を 2 つのセルが属すユニットから削除できます。

完成するループが有意であるように構築するための条件をまとめると以下のようになります。

Nice loop 連鎖ルール

1) Strong link はリンクを実線で、ラベルを「+数字」または単に「数字」形式で書く。
2) Weak link は、リンクを破線で、ラベルを「-数字」形式で書く。
3) strong inference は weak inference と解釈してもよい。
4) strong inference から strong inference にリンクを拡大するときはラベルが異なってい

なければならない。
5) weak inference から weak inference にリンクを拡大するときはラベルが異なっていなければならず、リンク元のノードは 2 値でなければならない。
6) strong inference から weak inference、またはその逆にリンクを拡大するときは「符号」の異なる同じ値のラベルであること。

Nice loop 例外条件

不連続 nice loop では開始セルに対してのみ nice loop 連鎖ルールが成立していなくてもよい。ただし、次のいずれかの条件が成立していること。

1) ループ開始セルが 2 つの weak link を持ち、それらのラベルが同じである。
2) ループ開始セルが 2 つの strong link を持ち、それらのラベルが同じである。
3) ループ開始セルが strong link と weak link を持ち、それらのラベルが異なる。

Nice loop が成立している場合、以下の定理が成立します

定理1:
連続的 nice loop 内の2つのリンクが共に strong inference であるセル X に対し、それらのリンクのラベルが A と B(A≠B)であるならば、セル X には A と B 以外の候補数字は存在できない。

定理2:
連続的 nice loop 内の2つのセル間のリンクが weak inference であるならば、そのリンクのラベルと同じ値の候補数字は、その2つのセルのどちらかに存在しなければならない。したがって、リンク・ラベルの表す候補数字を2つのセルが属すユニットから削除できる。

定理3:
不連続 nice loop 内の不連続点であるセル X に対する2つのリンクが共に strong inference でありラベルが共に A であるならば、セル X の値は A である。

定理4:
不連続 nice loop 内の不連続点であるセル X に対する2つのリンクが共に weak inference でありラベルが共に A であるならば、セル X から候補数字 A を削除できる。

定理5:
不連続 nice loop 内の不連続点であるセル X に対する2つのリンクが strong inference と weak inference であり、weak inference のラベルが A であるならば、セル X から候補数字 A を削除できる。

連続的 Nice Loop、または不連続 Nice Loop は、ひとつの盤面に対して典型的には 10 から 20 程度存在します。しかし実際に候補数字を削除できる Nice Loop は 1 個から 5 個程度であることが普通です。そして数字候補を削除できる Nice Loop の大半は不連続 Nice Loop です。

Bi-value セルと Bi-location セルを手掛かりに Nice Loop を構築する

Nice Loop では 2 値セルと 2 所セルを手掛かりとしてループを構築します。2 値セルというのは定義で述べているようにセル内の候補が 2 つだけに絞り込まれた状態のセルのことです。たとえばセルの内容が「5」と「6」だけに絞り込まれたセルは 2 値セルです。

2 つのセルが同じ行・列・ブロックのいずれかに属し、そのセル内の共通の候補が同じ行・列・ブロックの中で、その 2 つのセルにしか存在しないとき、それら 2 つのセルを共通の候補を介した 2 所セルと呼びます。たとえば 4 列目に存在するすべてのセルを調べた結果、候補数字「3」が 1 行目と 4 行目のセルにしか存在しなければ 1 行 4 列のセル R1C4 と 4 行 4 列のセル R4C4 は 3 を介した 2 所セルです（図 1 参照）。

共通の候補数字を持ち、同じ行・列・ブロックに存在する 2 値セル同士は weak link の定義を満たします。同じ行・列・ブロックに存在する 2 所セルは strong link の定義を満たします。何故ならばリンク関係にある（=2 つのセル A と B が同じユニットに属し、共通の数字候補を持つ）ならば、共通の候補数字が 2 つのセルの両方に同時に存在することは数独（ナンプレ）のルール上許されないからです。たとえば同じ行に存在する 3 を介した 2 所セルの両方に 3 が存在することはルール上あり得ません。

2 値でも 2 所でもなく更に確定値でもないセル同士には、weak link が存在する可能性があります。しかし Nice Loop を構成する際にすべての weak link が重要な訳ではなく、一定の基準を満たす weak link だけが役に立ちます。

このとき「b/b plot」(http://arxiv.org/abs/cs.DS/0507053)と呼ばれる手順にしたがってセル間のリンクを構成すると効率的に Nice Loop を探索できます。b/b plot の構築手順は以下の通りです。

b/b plot 構築手順

1) Strong link はリンクを実線で、ラベルを「+数字」または単に「数字」形式で書く
2) Weak link は、リンクを破線で、ラベルを「-数字」形式で書く。
3) ユニット内の二択関係にある 2 所(bilocation)ノード間を実線で結ぶ。
4) ユニット内の共通の数字候補を持つ 2 値(bivalue)ノード間を破線で結ぶ。
5) ユニット内の実線を持つノードのラベル同士が一致していたら、そのノード間を破線で結ぶ。
6) ユニット内の実線を持つノードからノードのラベルと同じ候補を持つ 2 値ノードへ破線を描く。
7) Nice Loop が成立しているか判定する。必要であれば破線を追加(手順 8 と 9 を参照)して Nice Loop を完成させる。
8) 実線を持つノードにノードのラベルと共通の候補数字を持つ同じユニット内の任意のノードから破線を追加しても良い。
9) 2 値ノードに対して共通の候補数字を持つ同じユニット内の任意のノードから破線を追加しても良い。

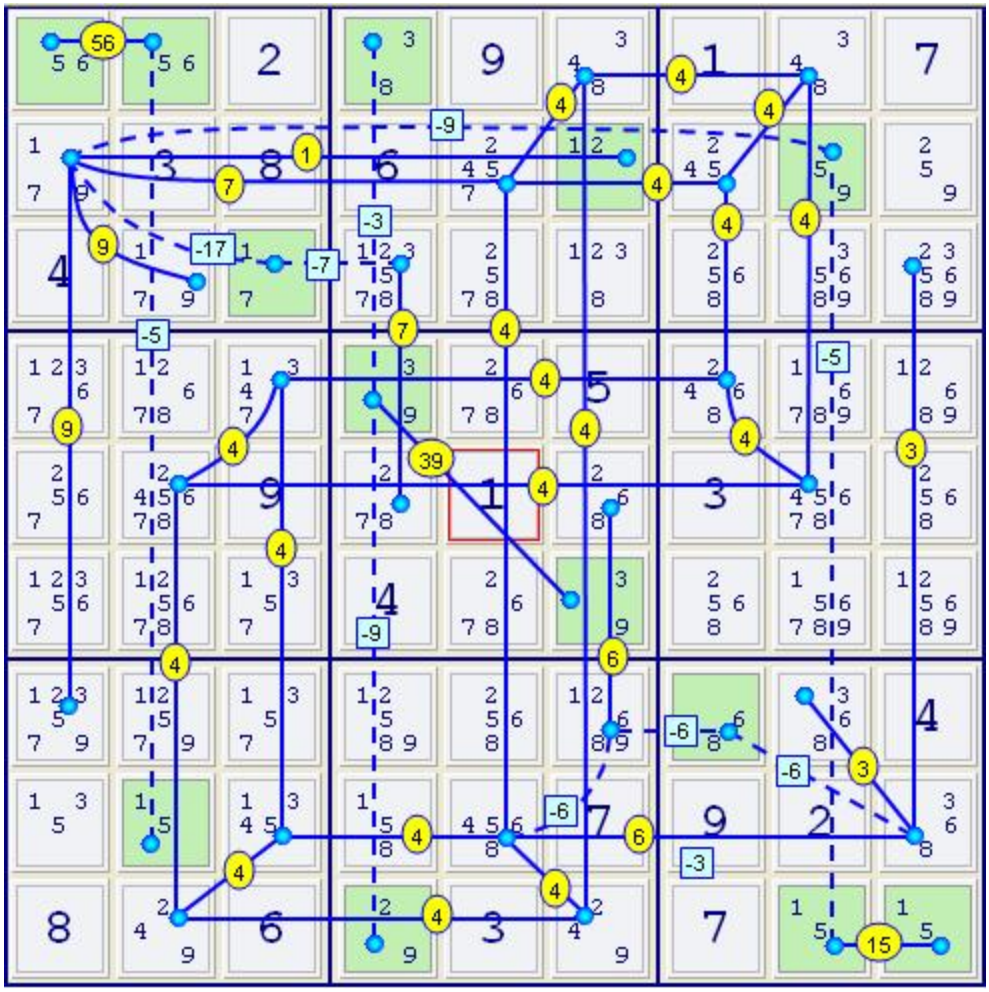

図１　b/b plotを行った例。手順8)と9)は行っていない。この図は「Sudoku Players' Forums」(http://www.sudoku.com/boards/viewtopic.php?t=2143)からの引用。

プログラムとしては、b/b plotによって構成したグラフ上のセルを開始点として前述のNice Loop連鎖ルールを満たす経路を探索しますが、人間がNice Loopを探索する場合はb/b plotを行う必要はなく、複数の候補を持つ任意のセルからNice Loop連鎖ルールを満たす経路の探索を始めて問題ありません。

Nice Loopを自身の手で発見しようとする際には、とにかく一段階ごとにNice Loop連鎖ルールを満たしているかを**厳重にチェックすることが大切**です。途中でミスが発生していると、それ以降のすべての作業が無駄になります。連鎖が長くなればなるほどミスが発生する確率が高くなります。連鎖数が３になったら連鎖開始セルとリンクできないかを常にチェックすることをお勧めします。

(Isao Daigo: mail2daigo@gmail.com)

参考リンク

[1] Nice loops for advanced level players - b/b plot
http://www.sudoku.com/boards/viewtopic.php?t=2143（at 2009/06/21）

[2] Nice Loop – Sudopedia
http://www.sudopedia.org/wiki/Nice_Loop（at 2009/06/21）

[3] Nice Loops in Sudoku
http://www.paulspages.co.uk/sudokuxp/howtosolve/niceloops.htm(2009/06/21)