1. 後ろ戻りアルゴリズム (*Backtrack search algorithms*)

		Go back →		
Go	↓	BT	BJ	CBJ
Forward	↓	BM	BMJ	BM-CBJ
	↓	FC	FC-BJ	FC-CBJ

BackTracking, BackJumping, BackMarking Conflict-directed, Forward Chaining

[文献] G. Kondrak and P. van Beek: A Theoretical Evaluation of Selected Backtracking Algorithms, *Artificial Intelligence*, **89**(1997), 365–387.

時間的な後ろ戻り (Chronological backtracking)：*label*

```
function  bt-label(i)
    for each v_k ∈ CD_i  do
    Set x_i = v_k  and consistent = true
    for j  from 1 to i − 1 do        /* 対既割当変数チェック */
        if ¬C_ij(x_i, x_j) then
            Remove v_k from CD_i  and set consistent = false
            Unassign x_i and break inner loop
        endif
    if consistent  then return (i + 1, true)
    endfor
    return (i, false)
end  bt-label
```

時間的後戻り (Chronological Backtracking)：*unlabel*

function *bt-unlabel*(i)

$h \;=\; i-1$ /* 直前に割当てた変数に戻る */

$CD_i \;=\; D_i$

Remove current value assigned to x_h from CD_h

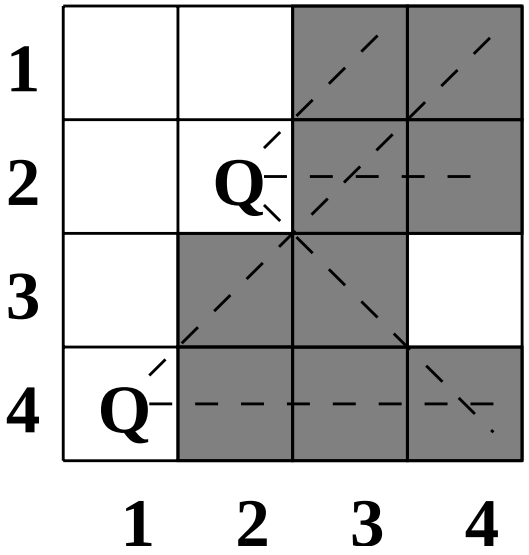
Unassign x_h

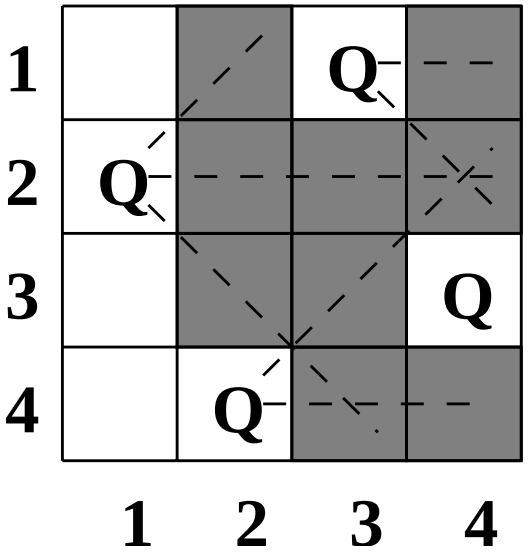
if CD_h is empty **then return** (h, false) /* 行止り */

else return (h, true) /* 次の値 */

end *bt-unlabel*

4 人の女王問題の部分解と完全解

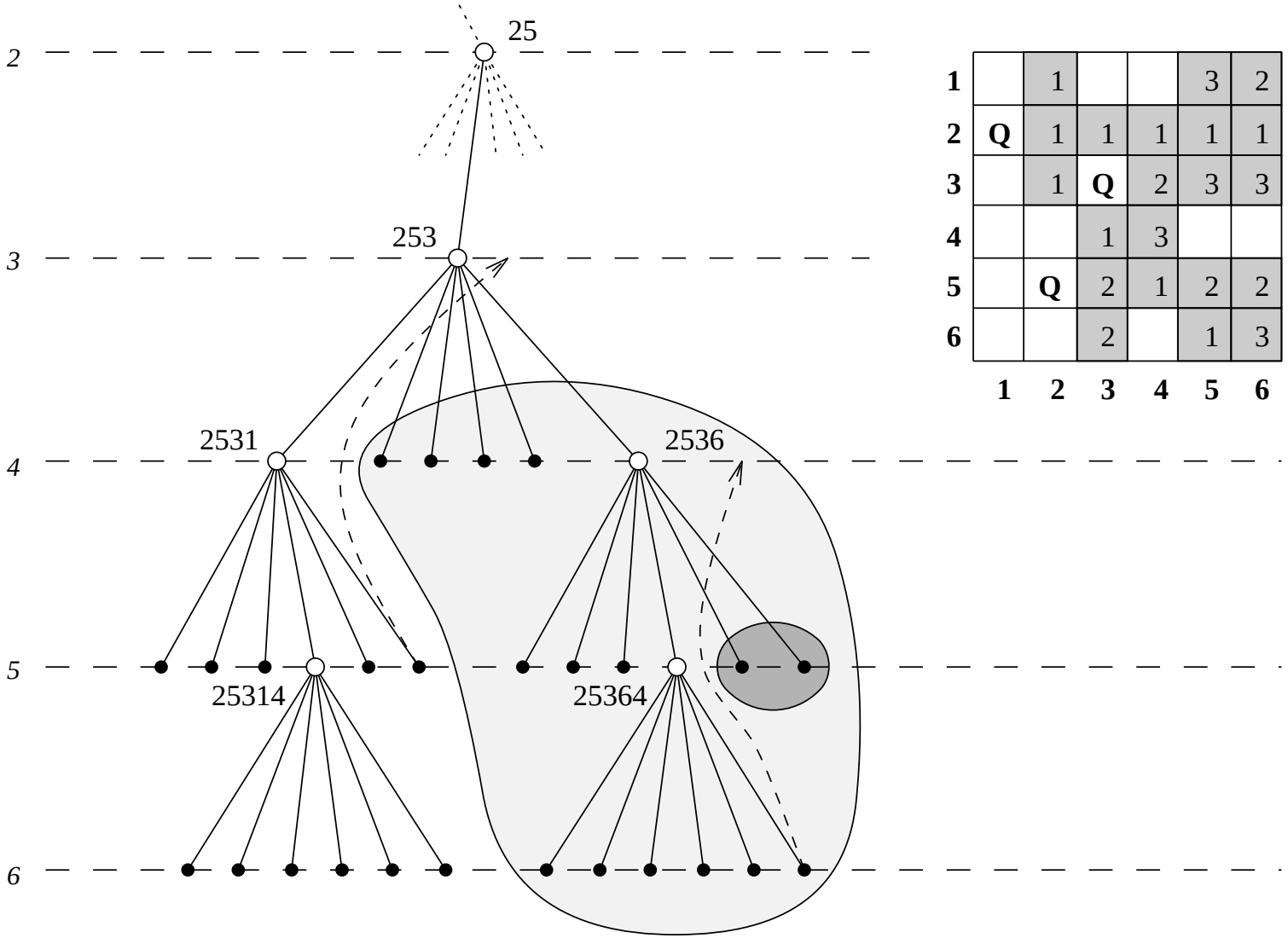




既に置かれた女王で取られる位置は影つき.

過去の変数, 現在の変数, 未来の変数 — 値の割当で定義

6 人の女王問題での BT 後ろ戻り木の一部 (BJ, CBJ)



1		1			3	2
2	**Q**	1	1	1	1	1
3		1	**Q**	2	3	3
4			1	3		
5		**Q**	2	1	2	2
6			2		1	3
	1	**2**	**3**	**4**	**5**	**6**

白点：無矛盾ノード，黒点：矛盾ノード．数字の並びは各行の選択列

Backjumping

$max\text{-}check_i$

- 無矛盾性チェックが行われた最深の変数を x_i とする
- x_i が無矛盾な値を持たない時，$max\text{-}check_i$ に一気に戻る．
- 初期値は 0

Backjumping : *label*

```
function  bj-label(i)
    for each v_k ∈ CD_i  do
    Set x_i = v_k  and consistent = true
    for j  from 1 to i − 1 do        /* 対既割当変数チェック */
        max-check_i = max(max-check_i, j)
        if ¬C_ij(x_i, x_j) then
            Remove v_k from CD_i  and set consistent = false
            Unassign x_i and break inner loop
        endif
    if consistent  then return (i + 1, true)
    endfor
    return (i, false)
end  bj-label
```

Backjumping : *unlabel*

function *bj-unlabel*(i)

h = $max\text{-}check_i$

for j **from** $h+1$ **to** i **do**

$max\text{-}check_j$ = 0

CD_j = D_j

endfor

Remove current value assigned to x_h from CD_h

Unassign x_h

if CD_h is empty **then return** $(h, false)$ /* 行止り */

else return $(h, true)$ /* 次の値 */

end *bj-unlabel*

Conflict-directed Backjumping (CBJ)

$conf\text{-}set_i$

- 変数を x_i と矛盾した過去の変数の集合
- 変数 x_h が x_i の値を排除するのに使われると，

  h を $conf\text{-}set_i$ に入れる
- 初期値は $\{0\}$

Conflict-directed Backjumping : *label*

```
function  cbj-label(i)
    for each v_k ∈ CD_i do
    Set x_i = v_k and consistent = true
    for j from 1 to i − 1 do        /* 対既割当変数チェック */
        if ¬C_ij(x_i, x_j) then
            conf-set_i = conf-set_i ∪ {j}
            Remove v_k from CD_i and set consistent = false
            Unassign x_i and break inner loop
        endif
    if consistent then return (i + 1, true)
    endfor
    return (i, false)
end  cbj-label
```

Conflict-directed backjumping : *unlabel*

function *cbj-unlabel*(i)

h = *max-list*(*conf-set*$_i$)

conf-set$_h$ = (*conf-set*$_h \cup$ *conf-set*$_i$) $\setminus \{h\}$

for j **from** $h+1$ **to** i **do**

conf-set$_j$ = {0}

CD_j = D_j

endfor

Remove current value assigned to x_h from CD_h

Unassign x_h

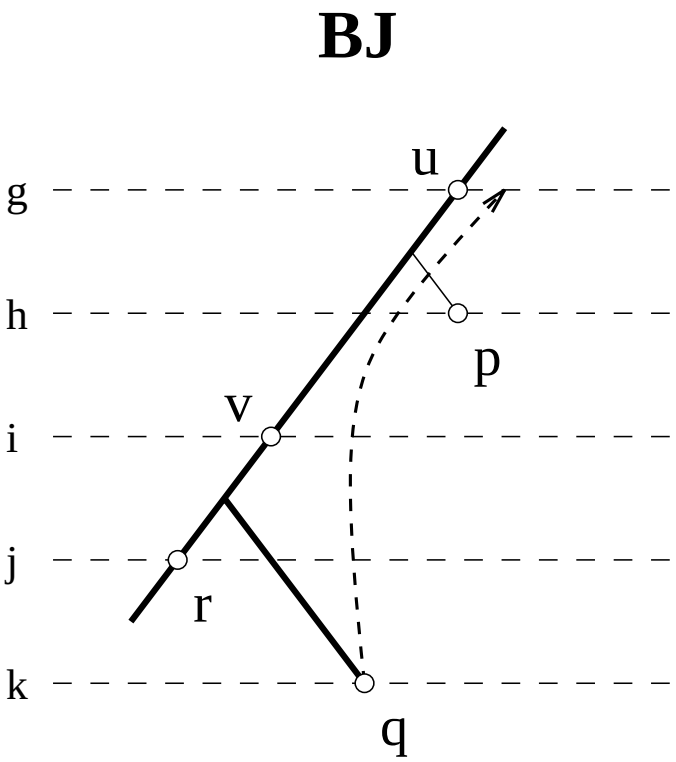
if CD_h is empty **then return** (h, *false*) /* 行止り */

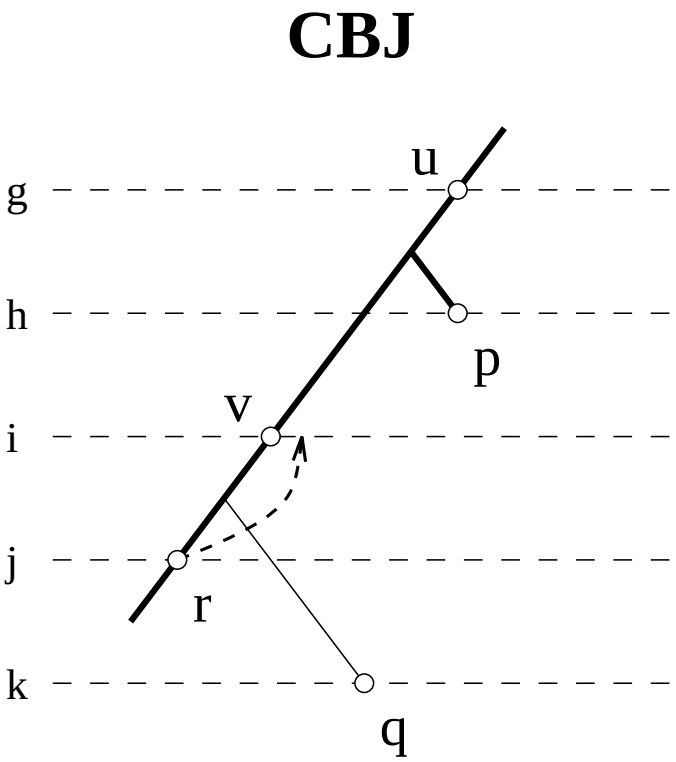
else return (h, *true*) /* 次の値 */

end *cbj-unlabel*

定理: BJ visits all nodes that CBJ visits.

帰謬法 (CBJ が訪問する p を BJ が訪問しない)





Backmarking

冗長な制約チェックを最小限にする

1. 変数 x_i に値 v が与えられる時に
   - v の変数 x_j に対する制約チェックが以前に失敗
   - そのチェック以降 x_j の値が不変

   $\Rightarrow$ 制約チェックも失敗することになる
2. 変数 x_i に値 v が与えられる時に
   - v の変数 x_j に対する制約チェックが以前に成功
   - そのチェック以降 x_j の値が不変

   $\Rightarrow$ x_j およびその後続ノードに対する制約チェックも成功することになる

Backmarking： データ構造

1. mcl_{ik} (初期値は 0)
   - 最大チェックレベルを示す $n \times d$ 配列
   - mcl_{ik} を $x_i = v_k$ がチェックされた最深変数
   - つまり，$max\text{-}check_i$ のより洗練版

   $\Rightarrow$ 変数 mcl_{ik} とその後続ノードに対する制約チェックはそれらの値が変化した時だけに行う
2. mbl_i (初期値は 0)
   - 最小バックアップレベル配列
   - x_i の現在の値割当以降に変化した最も浅い変数を記録

   $\Rightarrow$ 変数 mbl_i よりも以前の変数はその時から不変

Backmarking : *label*

```
function  bm-label(i)
  for each v_k ∈ CD_i do
    if mcl_ik ≥ mbl_i then
      Set x_i = v_k and consistent = true
      for j from mbl_i to i − 1 do
        mcl_ik = j
        if ¬C_ij(x_i, x_j) then
          Remove v_k from CD_i and consistent = false
          Unassign x_i and break inner loop
        endif
      endfor
      if consistent then return (i + 1, true)
    else Remove v_k from CD_i
  endfor
  return (i, false)
end  bm-label
```

Backmarking : *unlabel*

function *bm-unlabel*(i)

h = i-1

CD_i = D_i

mbl_i = h

for j **from** $h+1$ **to** n **do**

mbl_j = $\min(mbl_j, h)$

endfor

Remove current value assigned to x_h from CD_h

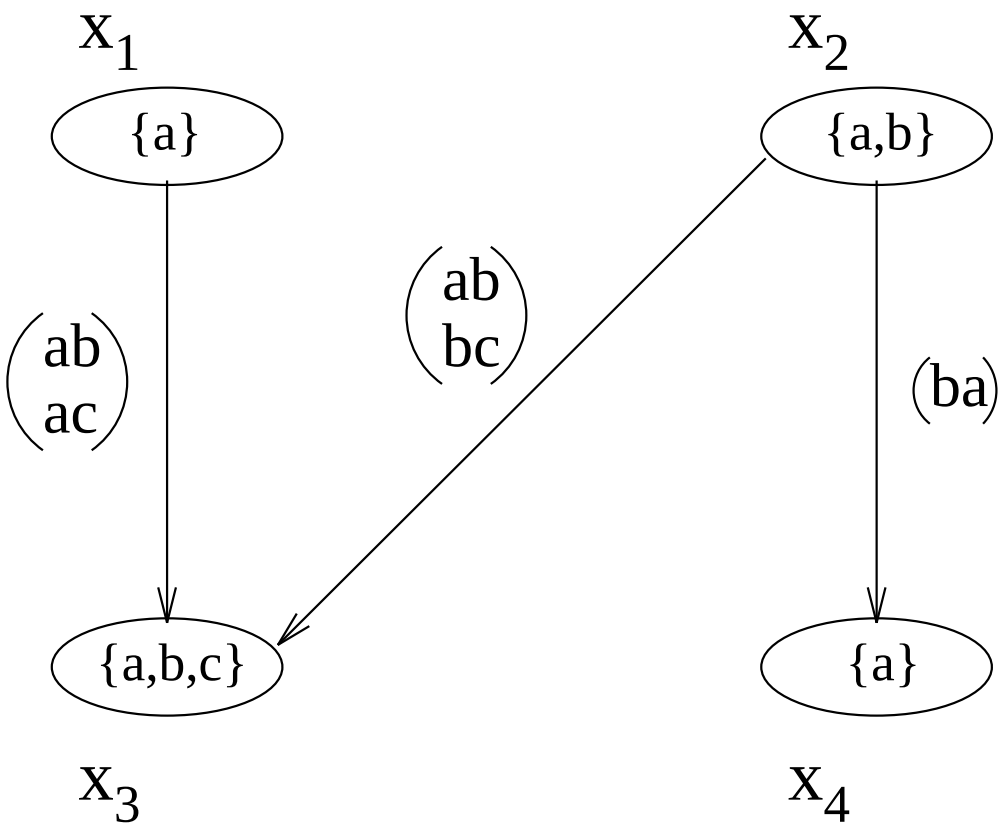
Unassign x_h

if CD_h is empty **then return** （h, $false$） /* 行止り */

else **return** （h, $true$） /* 次の値 */

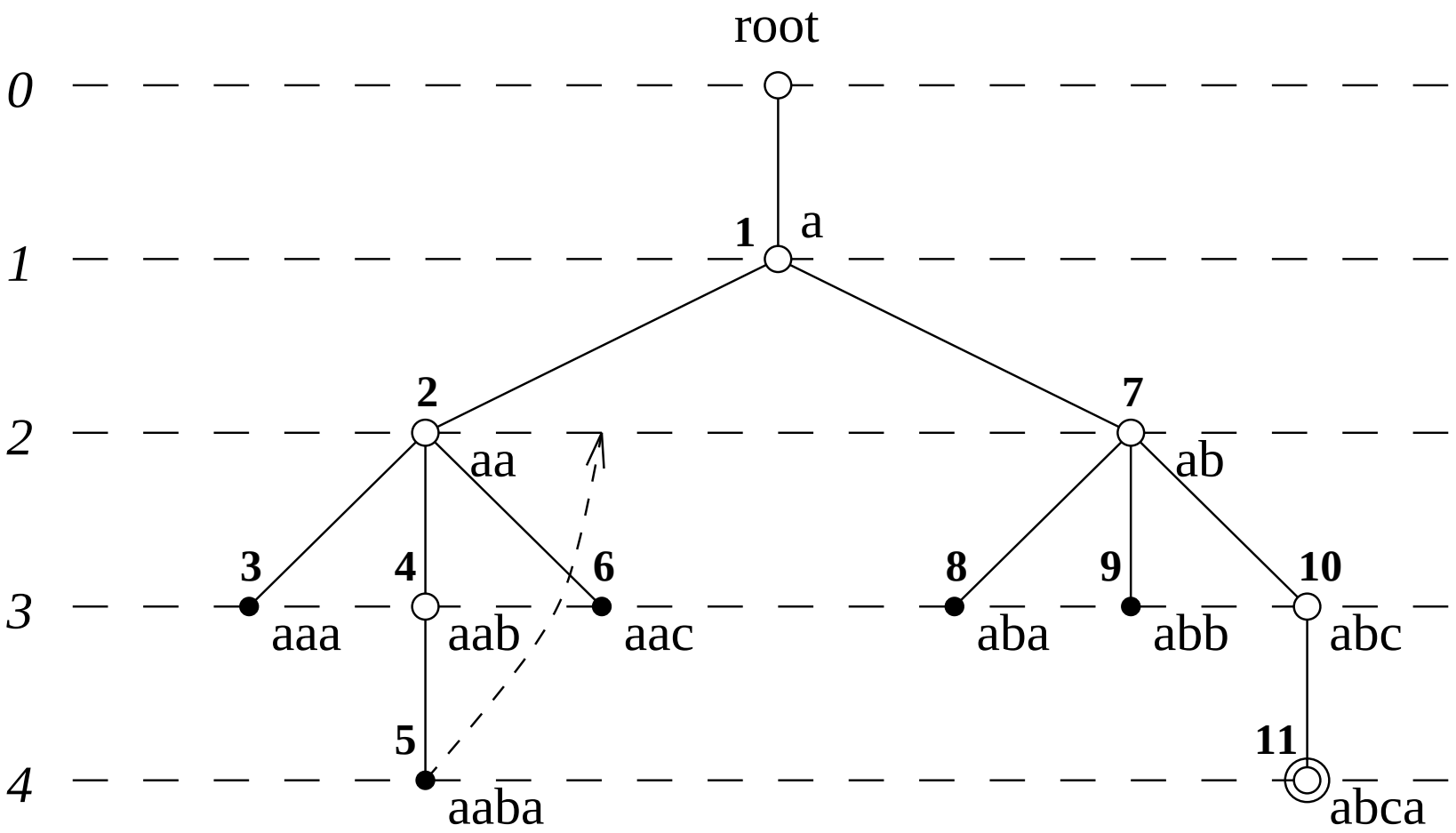
end *bm-unlabel*

簡単な制約充足問題



訪問する変数順序: x_1, x_2, x_3, x_4

選択木の構造



BT での制約チェック回数：

BM での制約チェック回数：

BMJ での制約チェック回数：

BM-CBJ：*label*

```
function  bm-cbj-label(i)
  for each v_k ∈ CD_i do
    if mcl_ik ≥ mbl_i then
      Set x_i = v_k and consistent = true
      for j from mbl_i to i − 1 do
        mcl_ik = j
        if ¬C_ij(x_i, x_j) then
          conf-set_i = conf-set_i ∪ {j}
          Remove v_k from CD_i and consistent = false
          Unassign x_i and break inner loop
        endif
      endfor
      if consistent then return (i + 1, true)
    else 削除 v_k from CD_i and conf-set_i=conf-set_i∪{mcl_ik}
  endfor
  return (i, false)
end  bm-cbj-label
```

BM-CBJ：*unlabel*

function $bm\text{-}cbj\text{-}unlabel(i)$

h = $max\text{-}list(conf\text{-}lset_i)$

$conf\text{-}set_h$ = $(conf\text{-}set_h \cup conf\text{-}set_i) \setminus \{h\}$

mbl_i = h

for j **from** $h+1$ **to** n **do**

mbl_j = $\min(mbl_j, h)$

$conf\text{-}set_j$ = $\{0\}$

CD_j = D_j

endfor

Remove current value assigned to x_h from CD_h

Unassign x_h

if CD_h is empty **then return** （h, $false$） /* 行止り */

else **return** （h, $true$） /* 次の値 */

end $bm\text{-}cbj\text{-}unlabel$

Forward checking

$x_i = v_k$ の試行の時, x_j $(i < j \leq n)$ の領域をフィルタ:

1. $reduction_j$ (初期値は $\{\}$)
   - 以前に与えられた値によって不許可となった D_j の値のリストのスタック
2. $past - fcj$ (初期値は $\{\}$)
   - x_j に対してチェックした過去変数のスタック
3. $future - fci$ (初期値は $\{\}$)
   - x_i に対してチェックした未来変数のスタック

Forward checkining : *label*

```
function  fc-label(i)
   for each v_k ∈ CD_i  do
         Set x_i = v_k  and  consistent = true
         for j from i+1  to  n do
            if ¬check-forward(i, j) then
               Remove v_k from CD_i and consistent = false
               undo-reductions(i)
               Unassign x_i and break inner loop
            endif
         endfor
         if consistent  then return (i+1, true)
   endfor
   return (i, false)
end  fc-label
```

Forward checking

function $check\text{-}forward(i,j)$

$reductions$ = $\{\}$

for each $v_k \in CD_i$ **do**

Set $x_i = v_k$

if $\neg C_{ij}(x_i, x_j)$ **then** $reductions$=$reductions \cup \{v_k\}$

endfor

if $reductions \neq \{\}$ **then**

CD_j = $CD_j \setminus reductions$

$push(j,\ future\text{-}fc_i)$

$push(reductions,\ reductions_j)$

$push(i,\ past\text{-}fc_j)$

endif

return $CD_j \neq \{\}$

end $check\text{-}forward$

Undo forward checking reductions

procedure $undo\text{-}reductions(i)$

  while $forward\text{-}fc_i$ is not empty **do**

    j = $pop(j, future\text{-}fc_i)$

    $reductions$ = $pop(reductions_j)$

    CD_j = $CD_j \cup reductions$

    $pop(past\text{-}fc_j)$

  endwhile

end $undo\text{-}reductions$

Forward Checking：*unlabel*

function *fc-unlabel*(i)

h = $i-1$

undo-reductions(h)

undate-current-domain(i)

Remove current value assigned to x_h from CD_h

Unassign x_h

if CD_h is empty **then return** $(h, false)$ /* 行止り */

else return $(h, true)$ /* 次の値 */

end *fc-unlabel*

Updating current domains

procedure $update\text{-}current\text{-}domain(i)$

$CD_i = D_i$

  for each $reductions \in reductions_i$ **do**

    $CD_i = CD_i \setminus reductions$

end $update\text{-}current\text{-}domain$

The Zebra Problem

There are five houses with five different colours, in each house lives a person of different nationalityhaving favorite drinks, cigaretes and pets, the information is:

- The *Englishman* lives in the *Red* house
- The *Spaniard* owns the *dog*
- The *Norwegian* lives in the *first house on the left*
- *Kools* are smoked in the *Yellow* hmbixouse
- The man who smokes *Chesterfields* lives in the *house next* to the man with the *fox*.
- The *Norwegian* lives next to the *Blue* house
- The *Winston* smoker owns *snails*.
- The *Lucky Strike* smoker drinks *orange juice*
- The *Ukrainian* drinks *tea*
- The *Japanese* smokes *Parliaments*
- *Kools* are smoked in the house *next to the house* where the *horse* is kept
- *Coffee* is drunk in the *Green* house
- The *Green* house is *immediately to the right* (your right) of the *Ivory* house item *Milk* is drunk in the *middle* house.

宿題 Zebra 問題を BJ, BM, FC などで解いて見よう.

Problem: Where does the *Zebra* live, and in which house do they drink *water* ?

House	Pet	Drink	Nationality	Cigaretts