

1. ノード整合性 (Node Consistency)
2. アーク整合性 (Arc Consistency)
3. パス整合性 (Path Consistency)
4. m 次整合性 (m -Consistency)

整合アルゴリズム

- AC-1
- AC-2 (Waltz の フィルタリング, 記号的弛緩法)
- AC-3
- AC-4
- AC-5

AC-1 : アーク整合アルゴリズム

procedure $AC-1(G)$

begin

$Q = \{(i, j) \mid (i, j) \in arc(G), i \neq j\}$

repeat

begin

$CHANGE = \text{false}$

for each $(i, j) \in Q$ **do**

$CHANGE = REVISE((i, j)) \vee CHANGE$

end

until $\neg CHANGE$

end $AC-1$

アルゴリズム *REVISE* — *ArcCons* とほぼ同じ

```
procedure REVISE((i, j))
begin
  DELETE = false
  for each  $v \in D_i$  do
    if  $C_{ij}(v, u)$  なる  $u \in D_j$  が存在しない then
      begin
        Remove( $x, D_i$ )
        DELETE = true
      end
  return DELETE
end REVISE
```

AC-1 の時間計算量

- どの制約に対しても *REVISE* は C_{ij} を $O(d^2)$ 回計算
- repeat 中で $|Q| = 2e$
- repeat での最悪のケースは, 1 回に 1 個しか値が削除されない場合.

つまり, repeat が d 回繰り返される.

⇒ AC-1 での最悪計算量は $O(ed^3)$

AC-2 : アーク整合アルゴリズム

```
procedure AC-2(G)
```

```
begin
```

```
  for  $i = 1$  to  $n$  do
```

```
     $Q = \{(i, j) \mid (i, j) \in \text{arc}(G), i < j\}$ 
```

```
     $Q' = \{(h, i) \mid (h, i) \in \text{arc}(G), h < i\}$ 
```

```
    while not EmptyStack( $Q$ ) do
```

```
      while not EmptyStack( $Q$ ) do
```

```
         $(k, m) = \text{pop}(Q)$ 
```

```
        if REVISE( $(k, m)$ ) then
```

```
          push( $Q'$ ,  $\{(p, k) \mid (p, k) \in \text{arc}(G), p \leq i, p \neq m\}$ )
```

```
        end
```

```
         $Q = Q'$  ;  $Q' = \text{empty}$ 
```

```
      end
```

```
    end
```

「人工知能特論」, 京都大学大学院情報学研究科知能情報学専攻, May 7, 2003 Lecture 3-5

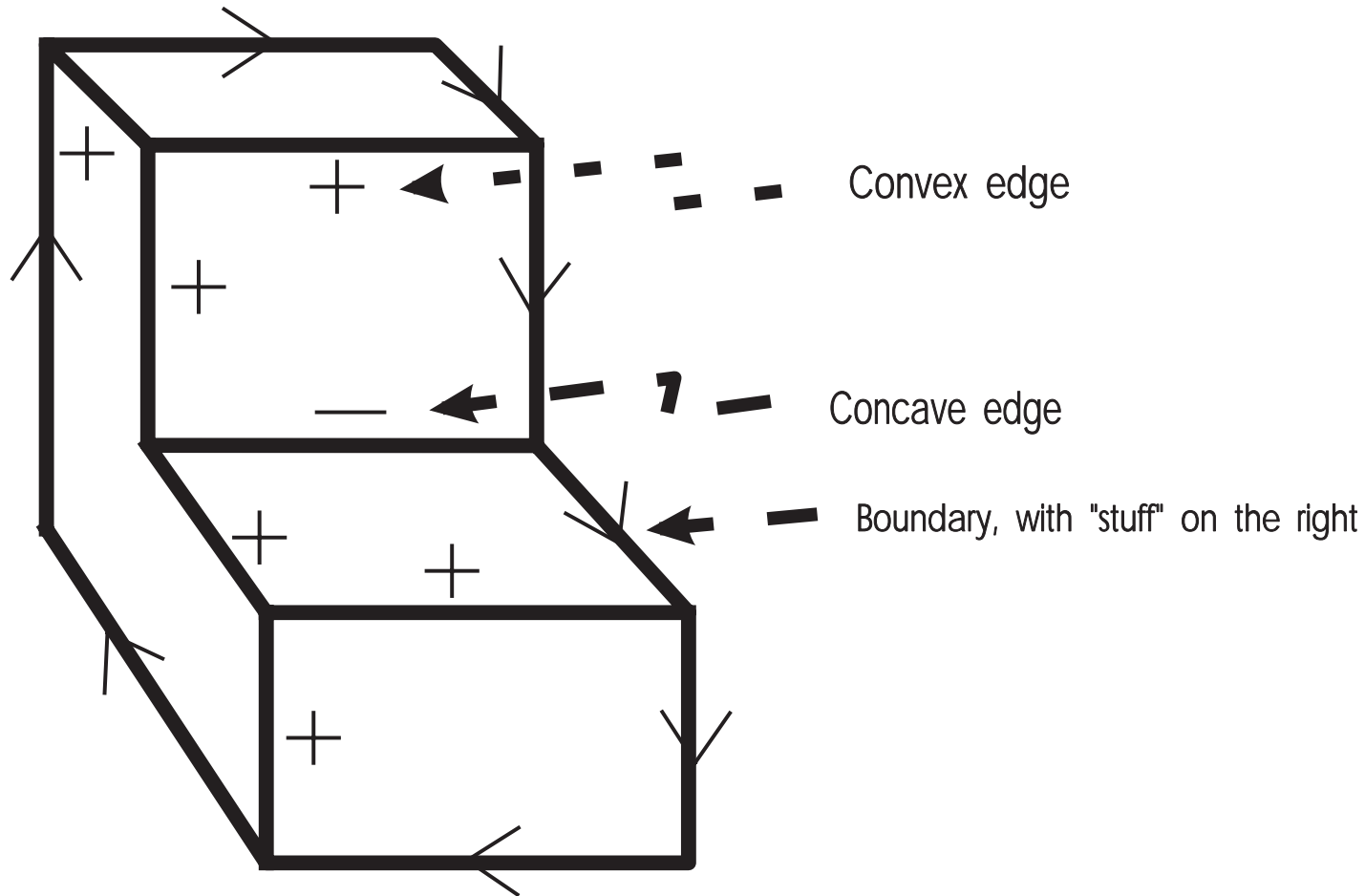
```
end AC-2
```

AC-2 の時間計算量

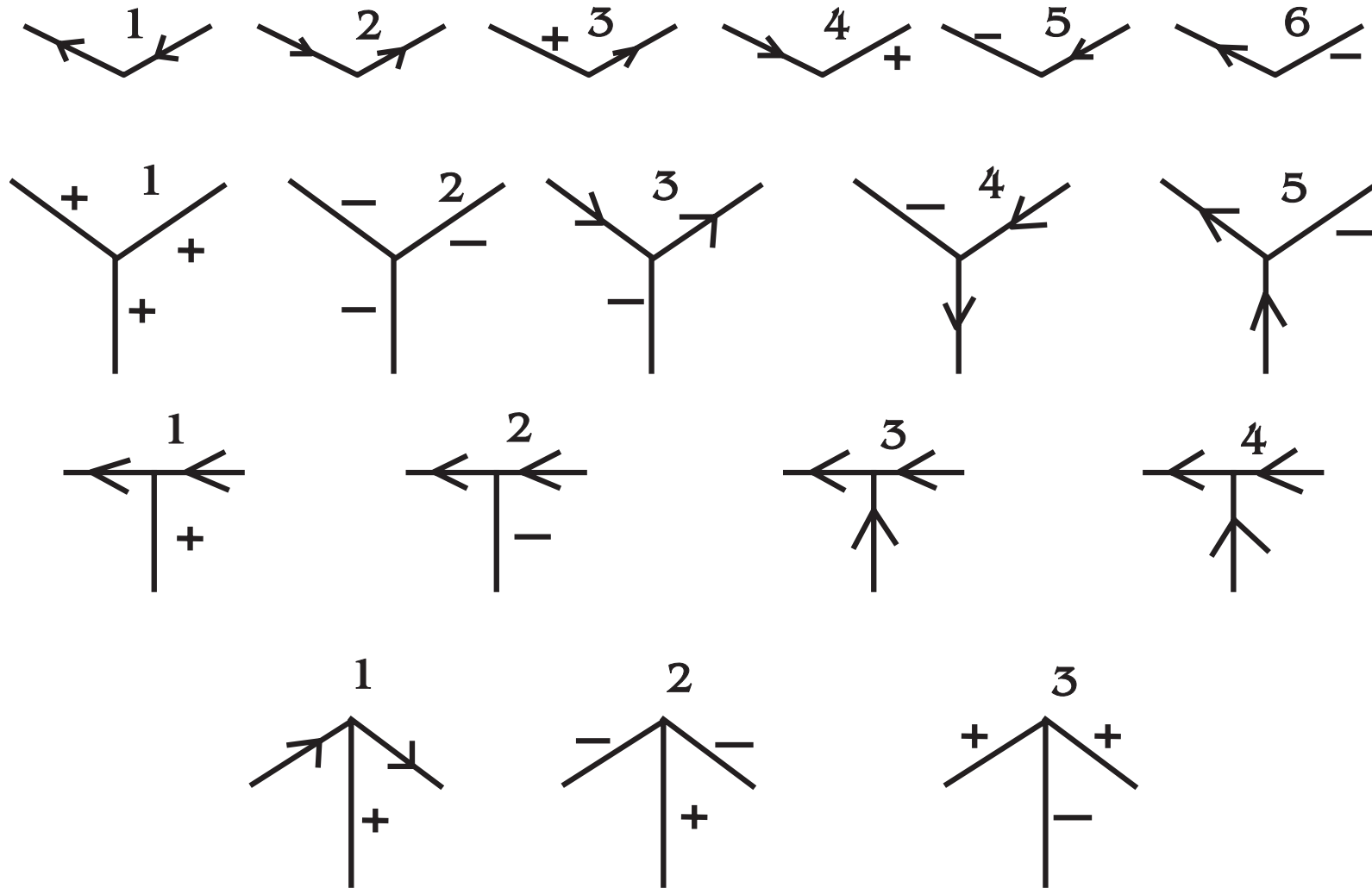
- AC-2 での最悪計算量は $O(ed^3)$
- Waltz の (フィルタリング) アルゴリズム
- 制約伝播法 (Constraint propagation) に基づいた
記号的弛緩法 (*symbolic relaxation method*) とも呼ばれる

記号的弛緩法 あるいは Waltz フィルタリング

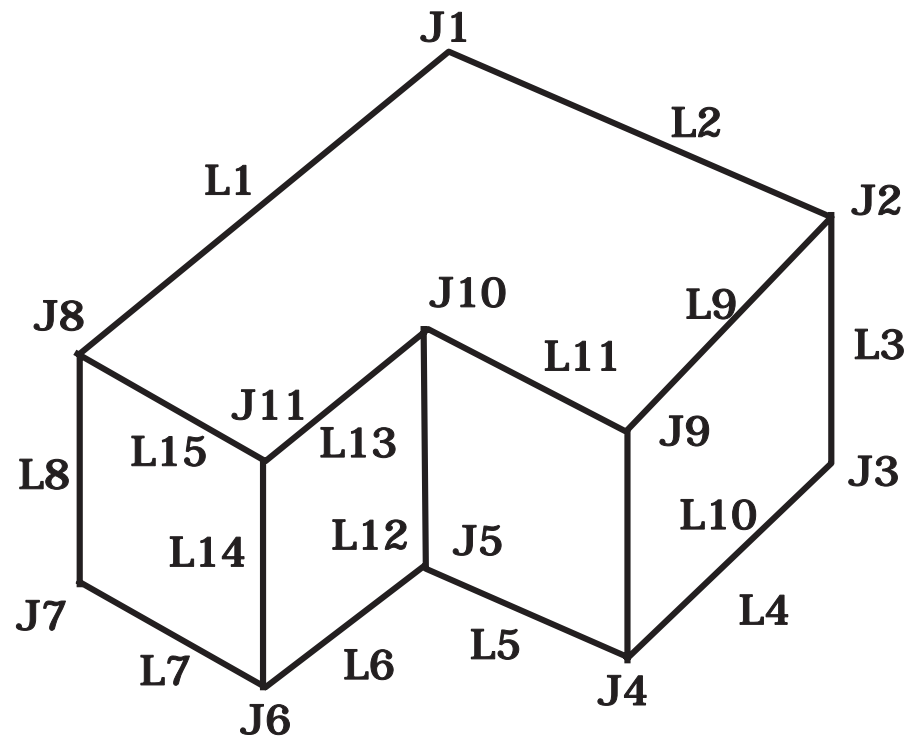
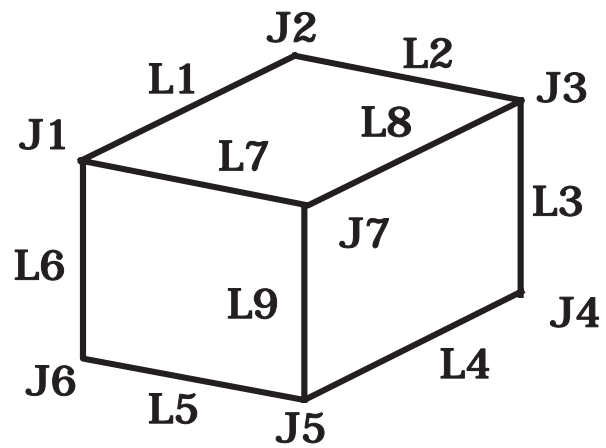
線画のラベル付け, 線ラベルの意味



単純な積木の世界のジャンクション・カタログ [Winston]

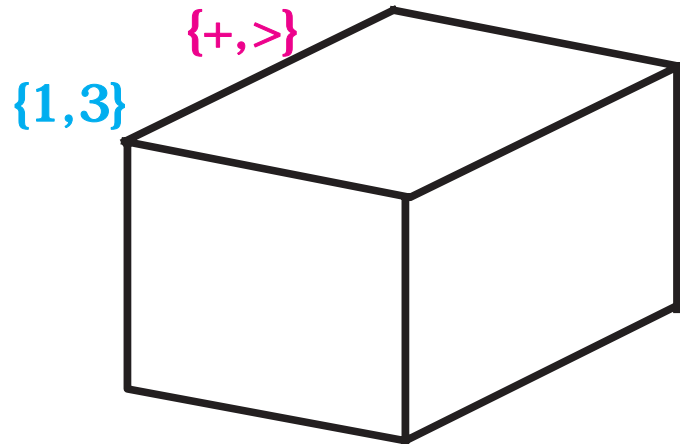


2つの例で解析を行う

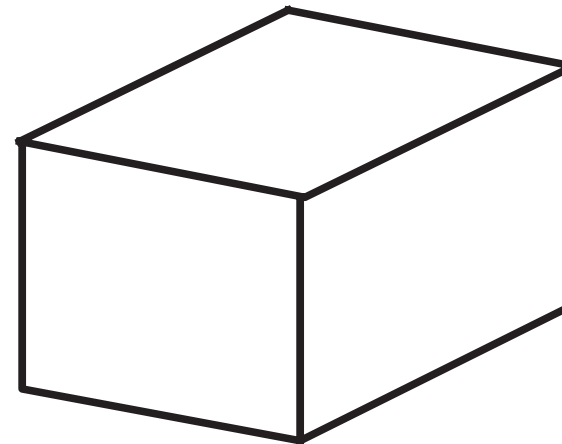


立方体の解析：可能性と無矛盾な解釈

可能なラベルつけ

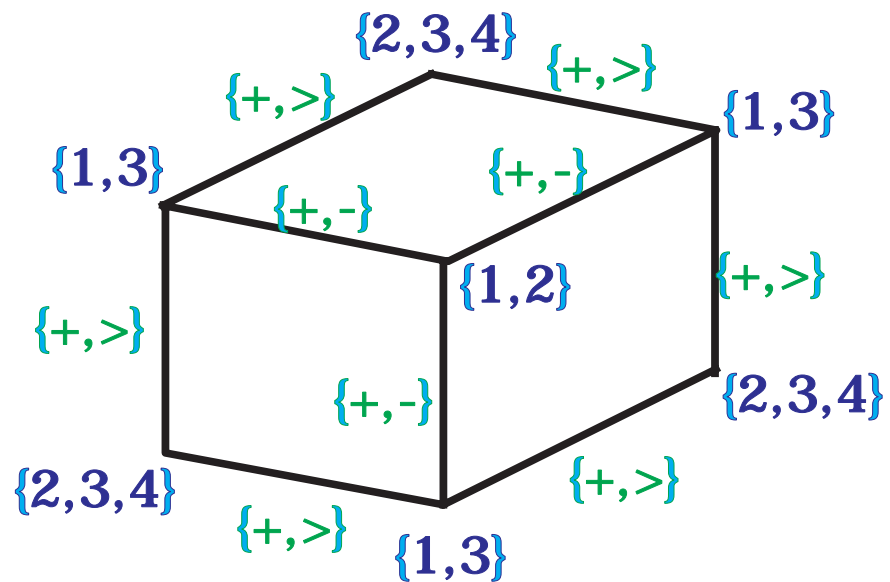


無矛盾な解釈

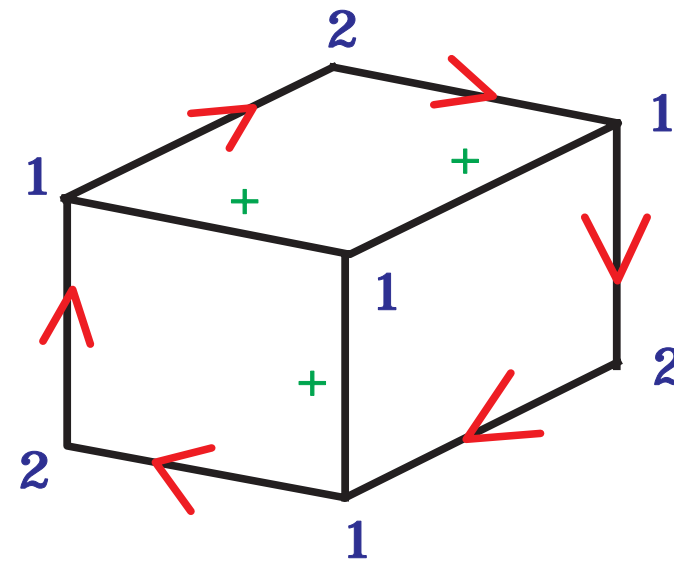


立方体の解析：可能性と無矛盾な解釈

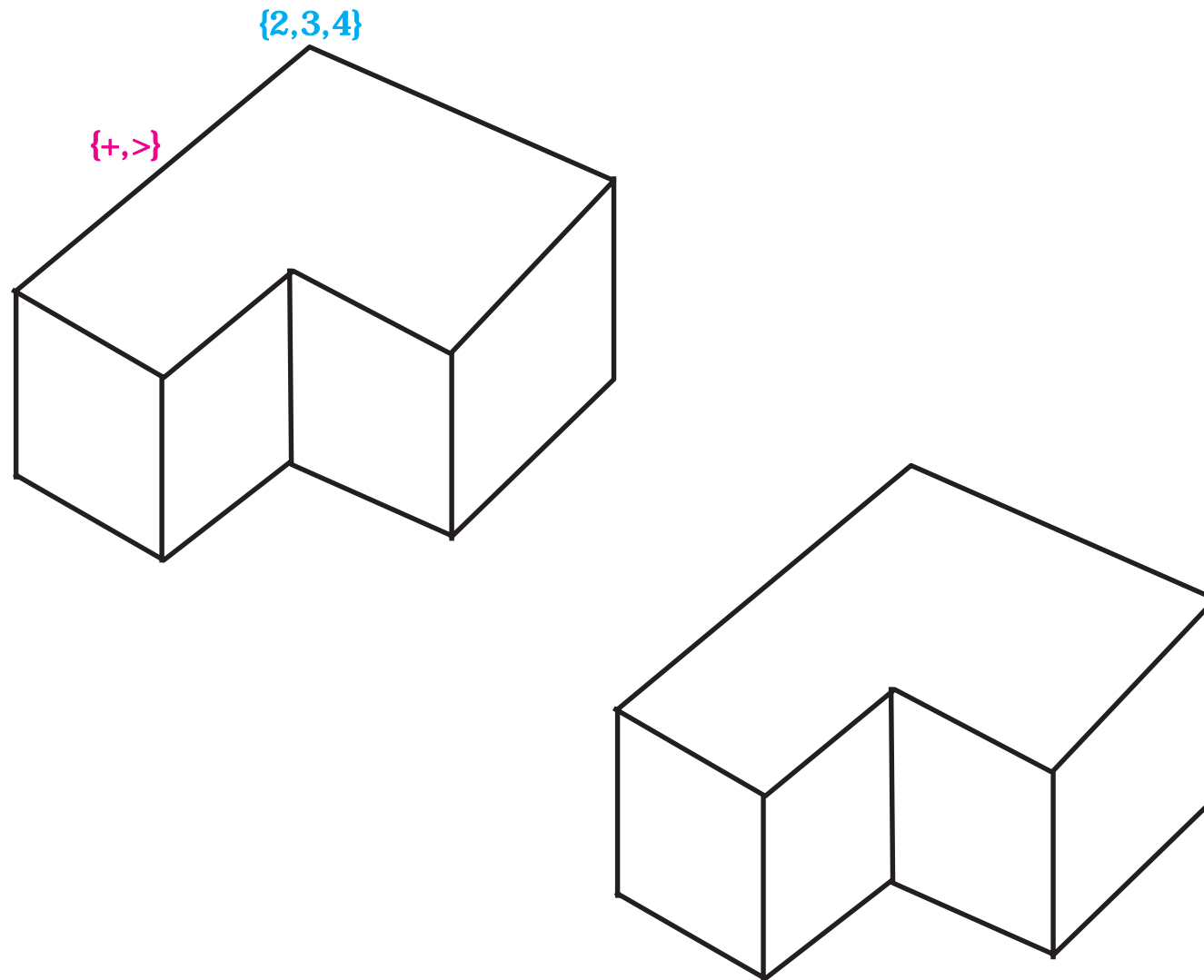
可能なラベルつけ



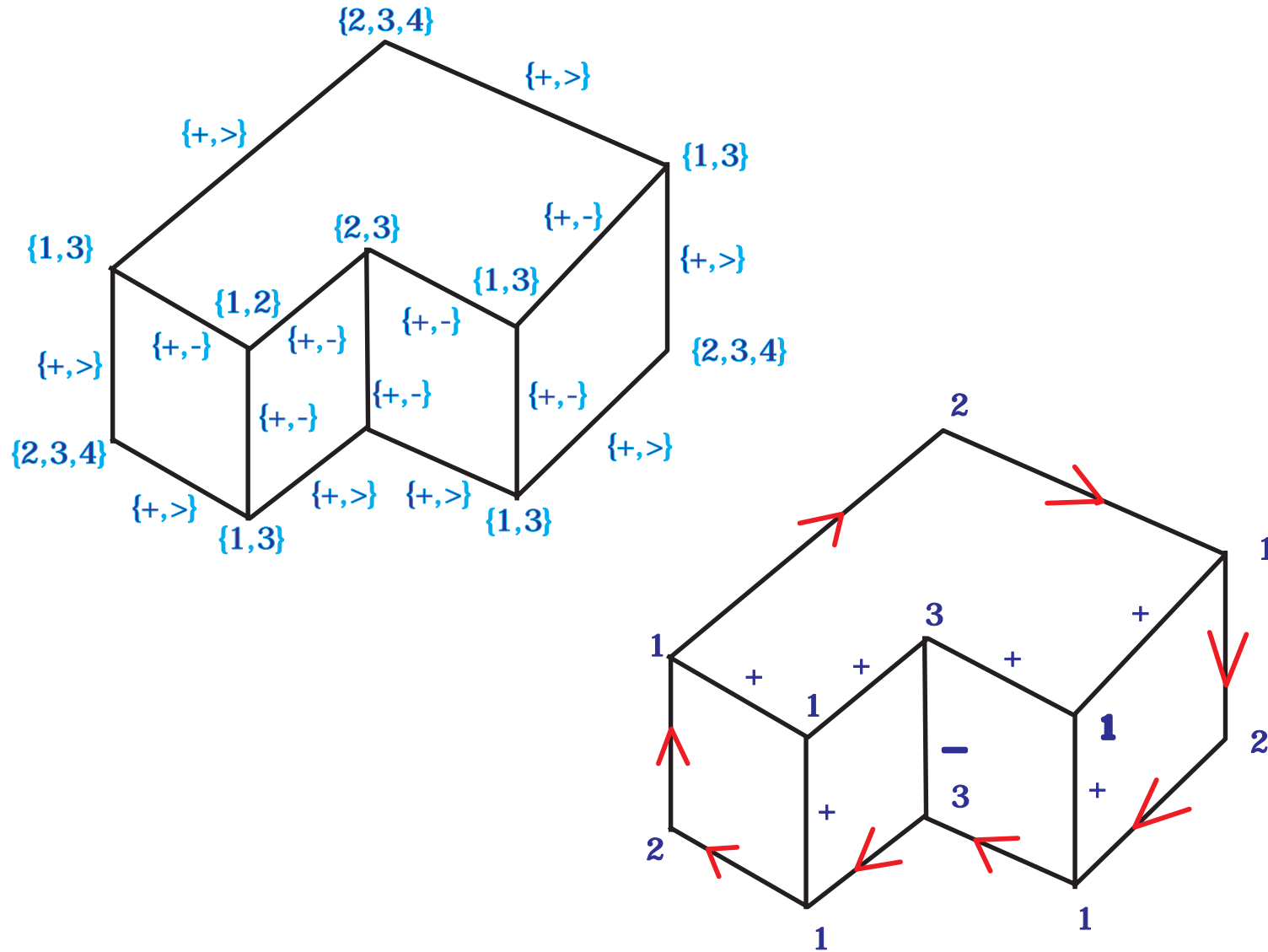
無矛盾な解釈



へこみのある物体の解析：可能性と無矛盾な解釈



へこみのある物体の解析：可能性と無矛盾な解釈



AC-3 : アーク整合アルゴリズム

procedure $AC-3(G)$

begin

$Q = \{(i, j) \mid (i, j) \in arc(G), i \neq j\}$

while not $EmptyQueue(Q)$ **do**

begin

$(k, m) = pop(Q)$

if $REVISE(k, m)$ **then**

$Q = Q \cup \{(i, k) \mid (i, k) \in arc(G), i \neq k, i \neq m\}$

end

end

end $AC-3$

AC-3 の時間計算量

- AC-2 : 深さ優先探索, AC-3 : 幅優先探索
- *REVISE* が成立する時 — ノード k にアークの個数 e_k とすると、高々 $e_k - 1$ 個アークが Q に追加.

$$\sum_{k=1}^n d(e_k - 1) = d(2e - n)$$

- repeat 中で少なくとも 1 個の値が除去
繰り返しの個数は Q の長さ, つまり, $2e + d(2e - n)$
- 各繰り返しでは, d^2 回の C_{ij} の評価が必要

⇒ AC-3 での最悪計算量は $O(ed^3)$

下限 : *REVISE* がすべて偽 ⇒ $\Omega(ed^2)$

AC-5 : アーク整合アルゴリズム

AC-5 は汎用 (*generic*) 整合アルゴリズム

- *ArcCons* と *LocalArcCons* という抽象化手続き使用
- AC-3 や AC-4 に特化可能
- 制約の特徴を活用するように特化可能
 - 関数型制約 (functional constraints)
 - 反関数型制約 (anti-functional constraints)
 - 単調型制約 (monotonic constraints)
 - 要素毎の関数型制約 (piecewise functional constraints)
 - 要素毎の反関数型制約 (piecewise anti-functional ...)
 - 要素毎の単調型制約 (piecewise monotonic constraints)

AC-5 の Queue 要素

AC-5 は $((i, j), w)$ という形式の *queue* を管理.

- (i, j) はアーク, w は D_j から取り除かれた値. 後でアーク (i, j) を再検討する時に使用.
- $Enqueue(j, \Delta, Q)$ は, アーク (i, j) の $w \in \Delta$ なる queue Q に $((i, j), w)$ 形式の全要素を挿入.

k

j C_{ij} を無矛盾化するために

D_j から除去する要素が Δ

i

ArcCons と LocalArcCons

function *ArcCons*(i, j)

Returns $\Delta = \{v \in D_i \mid \forall u \in D_j \neg C_{ij}(v, u)\}$

D_i から Δ の要素を除去し (i, j) を consistent にする

function *LocalArcCons*(i, j, w)

D_j から w が除去されていると仮定

Returns Δ s.t. $\Delta_2 \supseteq \Delta \supseteq \Delta_1$ where

$$\Delta_1 = \{v \in D_i \mid C_{ij} \wedge \forall u \in D_j \neg C_{i,j}(v, u)\}$$

$$\Delta_2 = \{v \in D_i \mid \forall u \in D_j \neg C_{i,j}(v, u)\}$$

AC-5 : アーク整合アルゴリズム

```
procedure AC-5( $G$ )  
  InitQueue( $Q$ )  
  for each  $(i, j) \in \text{arc}(G)$  do  
     $\Delta = \text{ArcCons}(i, j)$   
    Enqueue( $i, \Delta, Q$ )  
    Remove( $\Delta, D_i$ )  end  
  
  while not EmptyQueue( $Q$ ) do  
     $((i, j), w) = \text{Dequeue}(Q)$   
     $\Delta = \text{LocalArcCons}(i, j, w)$   
    Enqueue( $i, \Delta, Q$ )  
    Remove( $\Delta, D_i$ )  end  
  
end AC-5
```

AC-5 での queue 演算回数

- 対 “(edge, value)” 対して, *Status* を導入
 - *InitQueue* : $Status((k, i), v) = \textit{present}$ if $v \in D_i$
 $\textit{rejected}$ otherwise
 - *EnQueue* : 各 queue 要素の *Status* = *suspended*
 - *DeQueue* : 各 queue 要素の *Status* = *rejected*
 - AC-5 のループで保持される不変関係 : $Status((k, i), v)$
 - = *present* iff $v \in D_i$
 - = *suspended* iff $v \notin D_i$ かつ $((k, i), v)$ が Q にある
 - = *rejected* iff $v \notin D_i$ かつ $((k, i), v)$ が Q にない
- ⇒ AC-5 での enqueue と dequeue は高々 $O(ed)$

AC-3 : アーク整合アルゴリズム

- どの制約に対しても *ArcCons* は $O(d^2)$
- AC-3 : AC-5 と同じ.

ただし, *LocalArcCons* は *ArcCons* を用いて実現

⇒ AC-3 での enqueue と dequeue は $O(ed^3)$

AC-4 : アーク整合アルゴリズム

- $ArcCons:O(d^2)$, $LocalArcCons:O(d) \Rightarrow AC-5: O(ed^2)$

	i		j	
支	2	u	u	{ u } 支持
持	2	v	v	{ u, v } リスト
数	1	w	w	{ v, w }
edge 辺りの時間		$O(d^2)$		$O(d^2)$
edge 辺りの空間		$O(d)$		$O(d^2)$

- $LocalArcCons(i, j, w)$ は edge (i, j) に対して w の「支持リスト」で繰り返し、「支持数」を 1 減らす。
「支持数」が 0 になる値の集合として Δ を計算.

$\Rightarrow ArcCons:O(d^2)$, $LocalArcCons:O(d) \Rightarrow AC-4: O(ed^2)$

関数型制約 (functional constraints)

制約 C が領域 D に関して関数的とは、iff $\forall v \in D$ に対して
 $\{w \in D \mid C(v, w)\}$ なる w が高々 1 つ存在

function *ArcCons*(i, j)

$\Delta = \{\}$

for each $v \in D_i$ **bf do**

if $f_{ij}(v) \notin D_j$ **then** $\Delta = \Delta \cup \{v\}$

return Δ

end *ArcCons*

ArcCons は $O(d)$

LocalArcCons は $O(1)$

function *LocalArcCons*(i, j, w)

AC-5 は $O(ed)$

if $f_{ij}(v) \in D_j$ **then return** $\{f_{ij}(w)\}$

else return $\{\}$

end *LocalArcCons*

課題 3 クロスワードパズル

AC-3 以上のアーク整合アルゴリズムを使用して解きなさい。

1		2		3
■	■		■	
■	4		5	
6	■	7		
8				
	■	■		■

Word List

AFT	HEEL	HOSES
ALE	HIKE	SAILS
EEL	KEEL	LASER
LEE	LINE	SHEET
TIE	KNOT	STEER