抽象解釈とLattice(束)モデル

2021年5月22日

舟越　和己

はじめに(1)

■抽象解釈(Abstract Interpretation)の手法が提案されたのは
１９７０年代。パリのÉcole Normale Supérieure（高等師範学校）
のP. Cousot, R. Cousotなどによりその基礎が確立されました。

■抽象解釈の古典とも言える論文のタイトルは、
「Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Constraction or Approximation of Fixpoints（抽象解釈：不動点の構築や近似によるプログラムの静的解析のための統一束モデル）:Patrick. Cousot and Radhia Cousot [1977年] 」

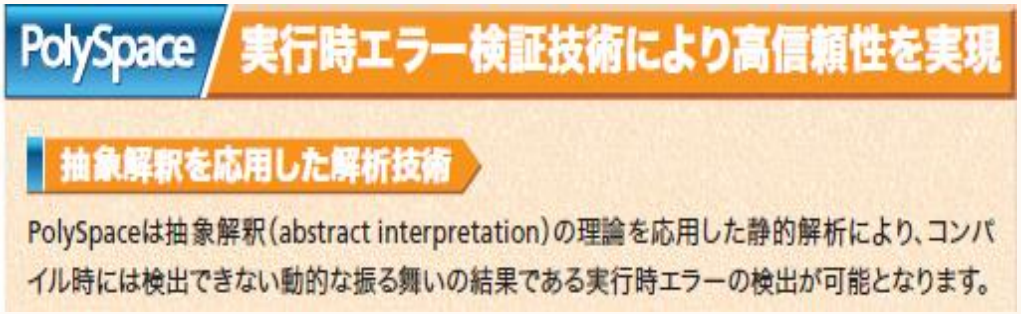
→このように、抽象解釈は数学の束論(Lattice Theory)に
基づく理論が適用されています。
また、不動点も抽象解釈のキーワードです。

はじめに(2)

■その後、仏のエコール・ノルマル等、主にヨーロッパで理論的研究が進み、現在、抽象解釈を応用した静的解析ツール等が商用化されています。

【ツール例】

・Mathworks社のPolyspace



・AbsInt社のツール群

→実行時間の上限値(最悪実行時間:WCET(*))を算出など

(*)WCET: Worst-Case Execution Time

本資料では、抽象解釈のLattice(束)モデル構造と、適用例である静的解析について簡単に説明します。

目次

1．準備：Lattice（束）の話から

・・・ここでは、抽象解釈の数学的基盤
について紹介します。

１．１　半順序集合(1)

２つの対象の順序関係は、日常の世界では、それらを大小関係(≦)などで一列に並べた時などに使われます（試験の点数順など）。

（例）整数の集合Zの中の数字の大小関係(≦)については、
下記のことが成り立ちます；
(i) a∈Z　ならば　a≦a　（反射律）
(ii) a, b, c∈Zのとき、a≦b かつ b≦c ならば a≦c　（推移律）
(iii) a, b∈Zのとき、a≦b かつ b≦a ならば a=b　（反対称律）
(iv) a, b∈Zのとき、a≦b または b≦a が成り立つ。（比較可能）

(i)から(iv)を満たす順序関係を「全順序」と言います。

ここで、(i)から(iv)の内、(iv)の条件が成り立たなくてもよい順序の世界を考え、数学ではそれを「半順序集合」と言います。

1．1　半順序集合(2)

■半順序集合の定義

集合L上の関係$\preceq$が半順序であるとは、下記(i)～(iii)を満たすことです；

(i)任意の$l \in L$ に対して、$l \preceq l$

(ii)任意の$l_1, l_2, l_3 \in L$ に対して、

$(l_1 \preceq l_2)$ かつ $(l_2 \preceq l_3) \Rightarrow l_1 \preceq l_3$

(iii)任意の$1_1, l_2 \in L$ に対して、

$(l_1 \preceq l_2)$ かつ $(l_2 \preceq l_1) \Rightarrow l_1 = l_2$

■半順序集合の例

Lを区間$[l, m]=\{x \in R \mid l \leqq x \leqq m,\ l, m \in Z\}$の集合とし、

半順序 $\preceq$を区間の包含関係（$\subseteq$）とします。

例えば、$[0, 1] \subseteq [0,2]$,　$[-1, 1] \subseteq [-2, 2]$など。

このとき、$(L, \subseteq)$は半順序集合となります。

１．２　半順序集合の上限･下限

■上限と下限の概念を区間の半順序集合の例で説明します；
(L, ⊆)を区間の半順序集合とするとき、
「2つの区間を含む区間」や「2つの区間に含まれる区間」を
考えることができます。
このとき、『「2つの区間を含む区間」の中で最小の区間』や
『「2つの区間に含まれる区間」の中で最大の区間』も考えられます。

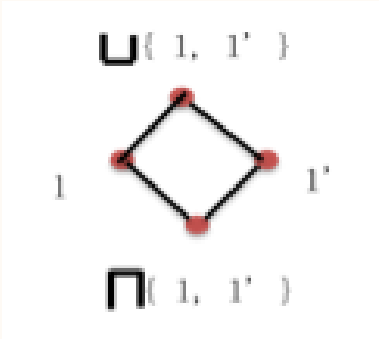
例１：{ [0, 1], [1,2] }を含む最小の区間=[0,2]
　　→これを、2つの区間[0, 1], [1,2]の上限といい、
　　⨆{[0, 1], [1,2]}で表します。

例２:{ [3, 5], [2,4] }に含まれる最大の区間=[3,4]
　　→これを、2つの区間[3,5], [2,4]の下限といい、
　　⨅{ [3,5], [2,4] }で表します。

１．３　Lattice（束）

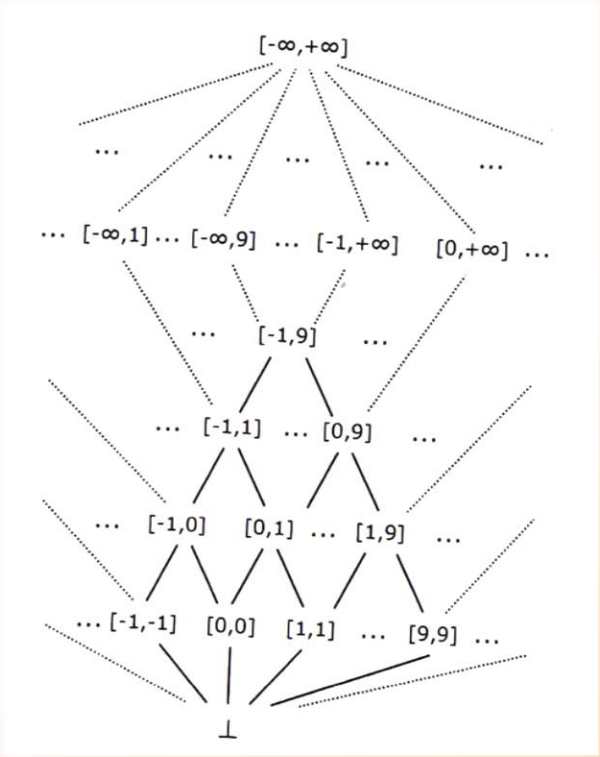
■半順序集合(L,$\preceq$)がLattice（束）であるとは、
任意のl, l'∈Lに対して、
その上限 $\bigsqcup\{ l, l' \}$ 及び、下限 $\bigsqcap\{l, l'\}$ が存在する
ことです。
・束の最小元=$\bigsqcap$L をボトムと言い、記号⊥で表します。
・束の最大元=$\bigsqcup$L をトップと言い、記号Tで表します。

l, l'と、それらの上限、下限を図示すると、次のような菱形に
なります。この菱形が繰り返されることにより、束はLattice状
になります。



1．4　完備束

■半順序集合(L, $\preceq$)が完備束であるとは、
任意の部分集合 K $\preceq$L に対して、
その上限$\bigsqcup$K 及び、
下限$\bigsqcap$Kが存在することです。



【完備束の例】: 区間の作る束
Zを整数の集合とするとき、Zを
Z‘=Z∪{−∞,∞}に拡張して、

$$\mathrm{Interval} = \{\bot\} \cup \{\, [z_1, z_2] \mid z_1 \leqq z_2,\ z_1 \in Z \cup \{-\infty\},\ z_2 \in Z \cup \{\infty\} \,\}$$

と定義すると、
Z’ 上の区間の束(Interval, ⊆)は完備束となります。

⊥は空(empty)の区間で、$[z_1, z_2]$ は区間の端点を含むz_1からz_2への区間です。

１．５　完備束の上で成り立つ性質：不動点定理

■単調写像
半順序集合(L, $\preceq$)から(L,$\preceq$)への写像 f: L→Lが単調写像であるとは、任意のl, l‘∈Lに対して、l$\preceq$ l’ならばf(l) $\preceq$ f(l ‘)となること。

■不動点
数学において写像の不動点とは、その写像によって自分自身に写される点のことである。
xが写像fの不動点であるとは、f(x)=xが成り立つときを言う。

完備束上の単調写像に対しては、次のタルスキの不動点定理が成立します。

■タルスキの不動点定理
完備束L上の単調写像は不動点を有する。

１．６　ガロア接続(1)

■ガロア接続は抽象解釈やモデル検査で対象の複雑さを単純化させる対応関係として使用されます。

「ガロア接続」を一言で説明すると、

一方の世界(L)で問題を解く代わりに、もう一方の世界(M)で問題を解くことができるような対応関係。
（Mの方がLより問題解決が容易な場合に有効）

１．６　ガロア接続(2)

■ガロア接続の定義

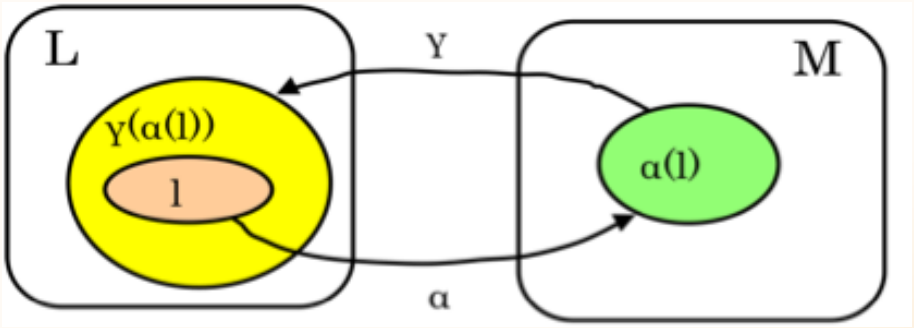
L=(L,⪯), M=(M,⪯)を完備束、l∈L, m∈Mとするとき、
α :L→M と γ :M→Lが単調写像ならば、
その写像の組み(α 、γ)が「ガロア接続」であるとは、
下記の条件を満たすことです;

$l \preceq \gamma(\alpha(l))$　　・・・・(1)

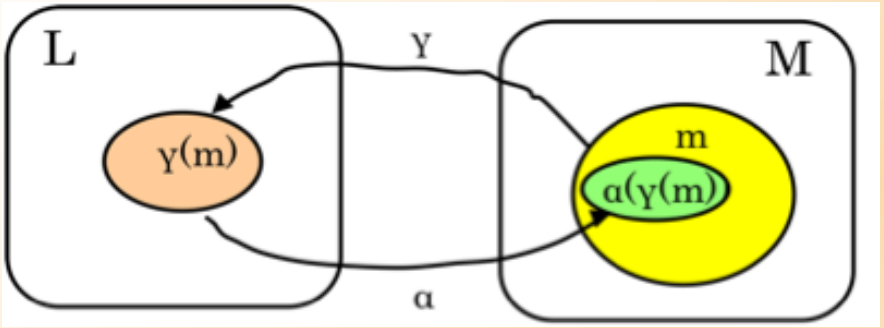
$\alpha(\gamma(m)) \preceq m$　　・・・・(2)

半順序(⪯)を包含(⊆)のイメージで (1), (2) を図示すると、

(1)



(2)

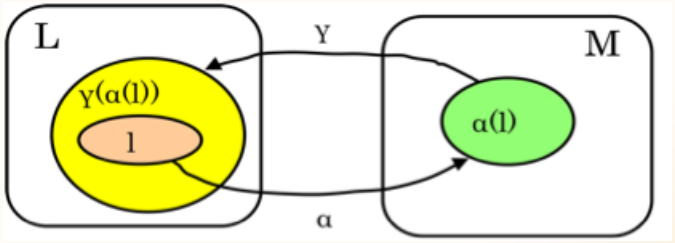


１．６　ガロア接続(3)

■ガロア接続の意味すること

・条件(1)　$l \preceq \gamma(\alpha(l))$　について：

この式は、「l∈Lをαで移したもの$\alpha(l)$はγでLの世界に戻すと、lより広くなる」ことを示しています。



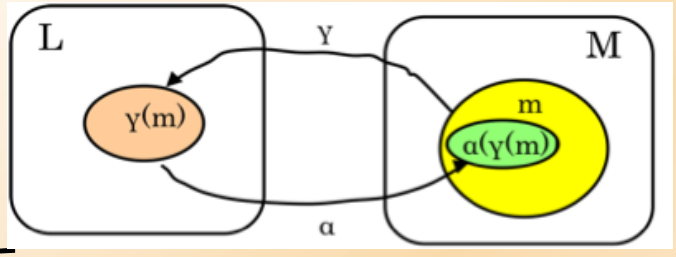
すなわち、α、γという対応関係により、

$\alpha(l)$は lより広い範囲を示しているので、$\alpha(l)$はlの近似です。

この意味からαは抽象化写像と言われます。

・条件(2)　$\alpha(\gamma(m)) \preceq m$　について：

この式は、「m∈Mをγで移したもの$\gamma(m)$はαでMの世界に戻すと、mより狭くなる」ことを示しています。



すなわち、α、γという対応関係により、

$\gamma(m)$はmより狭い範囲を示しています。

この意味からγは具体化写像と言われます。

1．7　ガロア接続の例(1)

■変数の数が1個ならば、区間を使用します。

Cを整数全体Zの部分集合とするとき、写像 α を

$$\alpha(C)=[\min(C), \max(C)]$$

I=[a, b]を区間とするとき、写像 γ を

$$\gamma(I)=\{ k\in Z \mid k\in[a, b] \}$$

とすると、(α 、γ)はガロア接続となります。

1．7　ガロア接続の例(2)

（例）C={ 1, 3, 4 }とすると、α(C)=[1, 4]

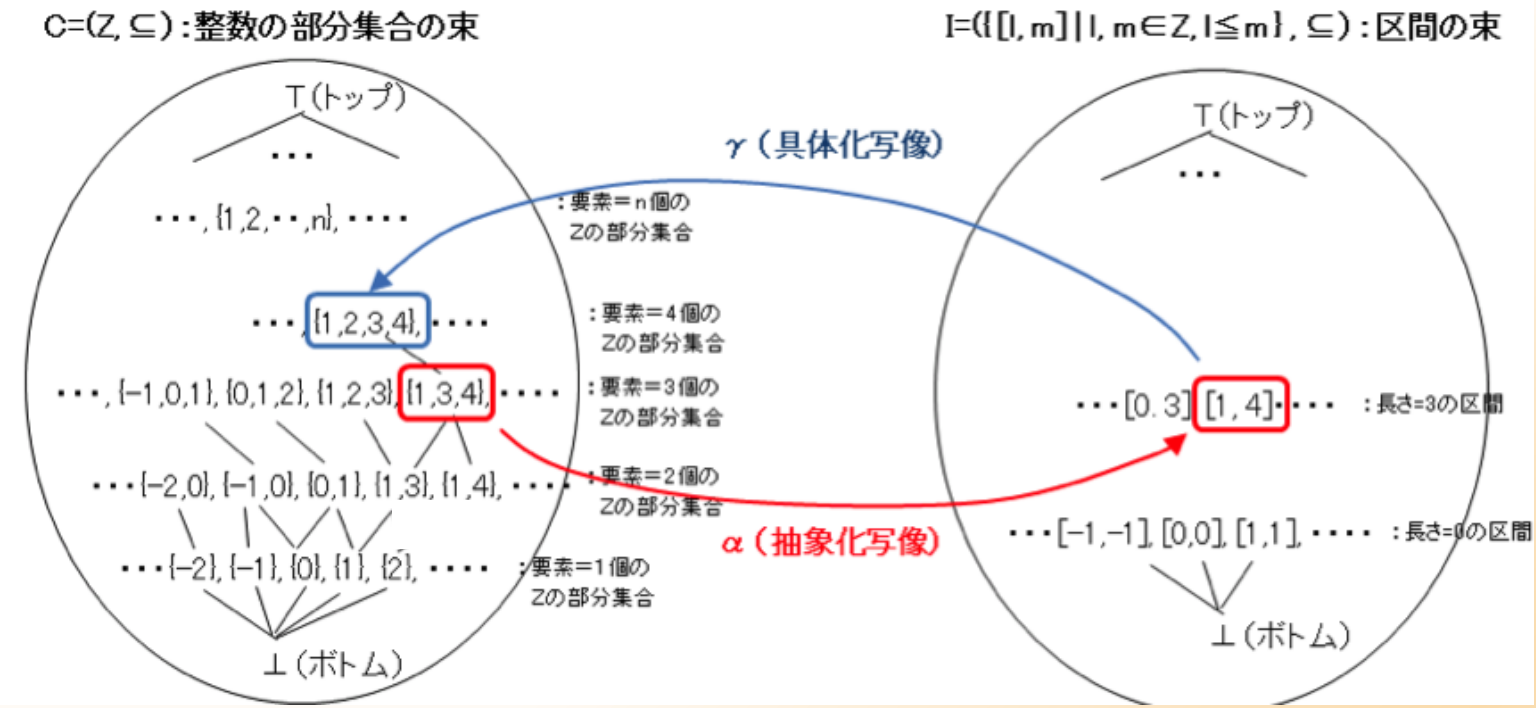
　　I=[1, 4]とすると、γ(I)={ 1, 2, 3, 4 }

→このとき、

　　C={ 1, 3, 4 }⊆ γ(α(C))={ 1, 2, 3, 4 }　・・・(1)

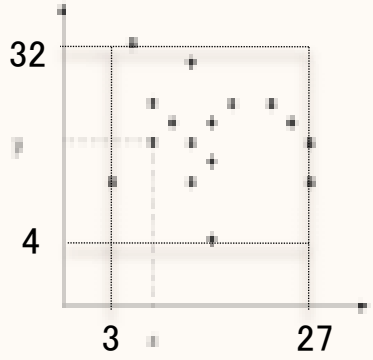
が成立します。また、

　　α(γ(I))=[1, 4] ⊆I　・・・(2)



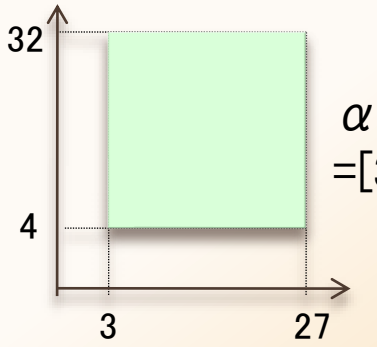
１．７　ガロア接続の例(3)

■前頁の例は変数が１個の場合でしたが、今度は変数がx, yの２次元になった場合を考えます（n変数の場合も同様）。
例えば、変数の値の集合Sが下の左図のような点に分布したとします。このとき、
α(S)の最も単純な例は「区間の直積」です（下の右図）。
但し、これは近似領域としては非常に粗い領域です。



S={…,⟨5, 7⟩,…,
⟨13, 21⟩,…}

α



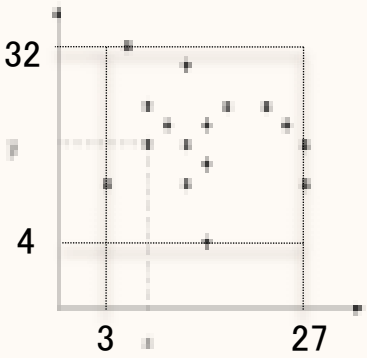
α(S)
=[3, 27]×[4, 32]

Fig. 1. Set of points

このとき、γ(α(S))= α(S)に含まれる全ての点の集合となります。

1．7　ガロア接続の例(4)

■より正確な近似化の例には「凸多面体」による近似があります。また、近似の精度は凸多面体よりもやや低下しますが、「Octagon（八面体）」近似もよく使われます。

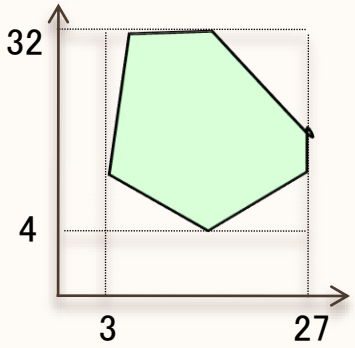


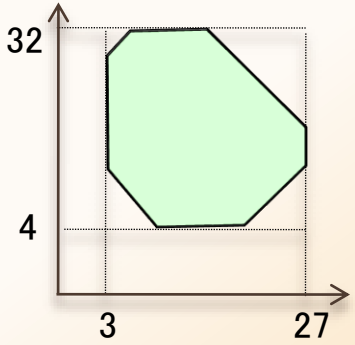
S= {…,⟨5, 7⟩,…,
⟨13, 21⟩,…}

Fig. 1. Set of points

α

α’





凸多面体
α(S)=x, yの
一次不等式群で
囲まれた領域

Octagon（八面体）
α(S)=(±x±y≦c,
±x≦d, ±y≦e)
という形式の一次
不等式群で囲まれ
た領域

このとき、γ(α(S))=α(S)に含まれる全ての点の集合となります。

2. 抽象解釈の考え方

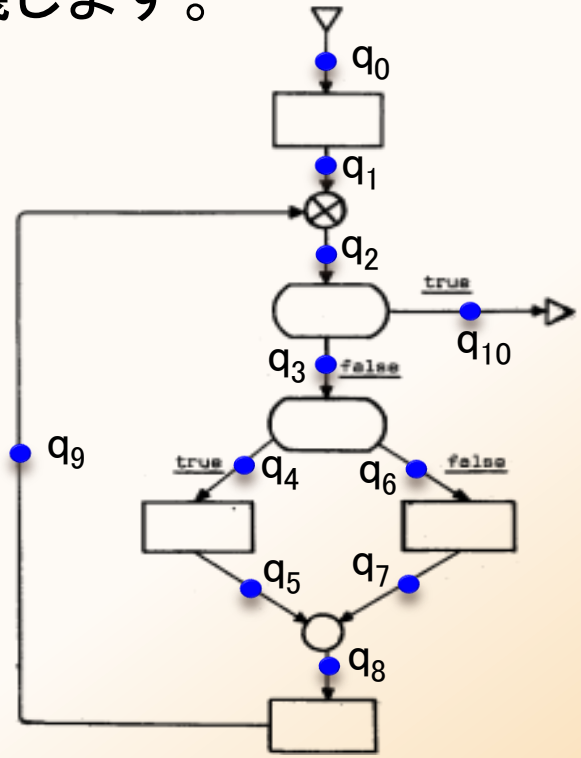
・・・ここでは、プログラムにおけるLattice（束）モデル
について紹介します。

２．１　プログラムポイント

最初に、用語「プログラムポイント(pt)」を定義します。

右図のように、プログラムは
- 入力ノード
- 代入ノード
- テストノード
- 単純な接合ノード
- ループ接合ノード
- 出口ノード

から構成されているとします。



「プログラムポイント(pt)」とは、隣接する
ノードを結ぶ矢印（弧）の場所を指します。
右図では、q_0 , q_1 , ・・・, q_{10}がプログラムポイントです。

２．２　プログラムに対する抽象解釈の着眼点(1)

■プログラムに対する抽象解釈の着眼点は、

①プログラムを実行したときのプログラム変数の値の集合

に着目し、

②各プログラムポイント(pt)において、

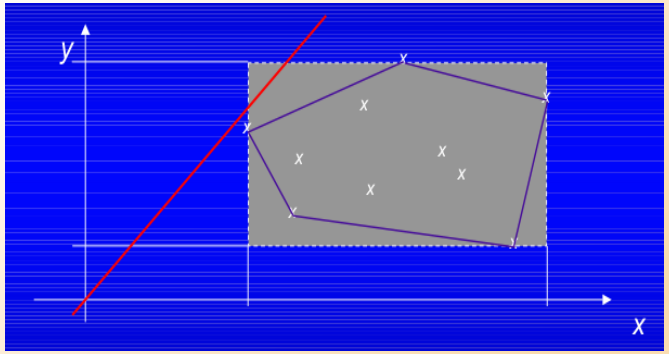
プログラムの変数が取り得る全ての値を含む集合を、
プログラムを実行することなく(すなわち、静的に)
求めることができれば
プログラム解析に利用できる

ということです。

２．２　プログラムに対する抽象解釈の着眼点(2)

例えば、「.../(x−y)というzero divideの可能性のある式を含むプログラム」をチェックする場合を考えます。

プログラムの変数x, yの値の分布（図では白色の×の記号）が下のグラフ表示のようになったとすると、抽象解釈では、
「プログラムの変数の個々の値（白色の×の記号）を考えるのではなく、全ての取り得る値を含む領域（図では多角形の領域）」
を考えます。
そして、この領域がプログラム変数の存在を禁止されているゾーン（図ではzero divideを示す赤い直線y=x）と交差していないかをチェックすることにより、プログラムの検証を行うという考え方です。
この図の場合、交差していないので確実にzero divideは発生しないことが保証されます。



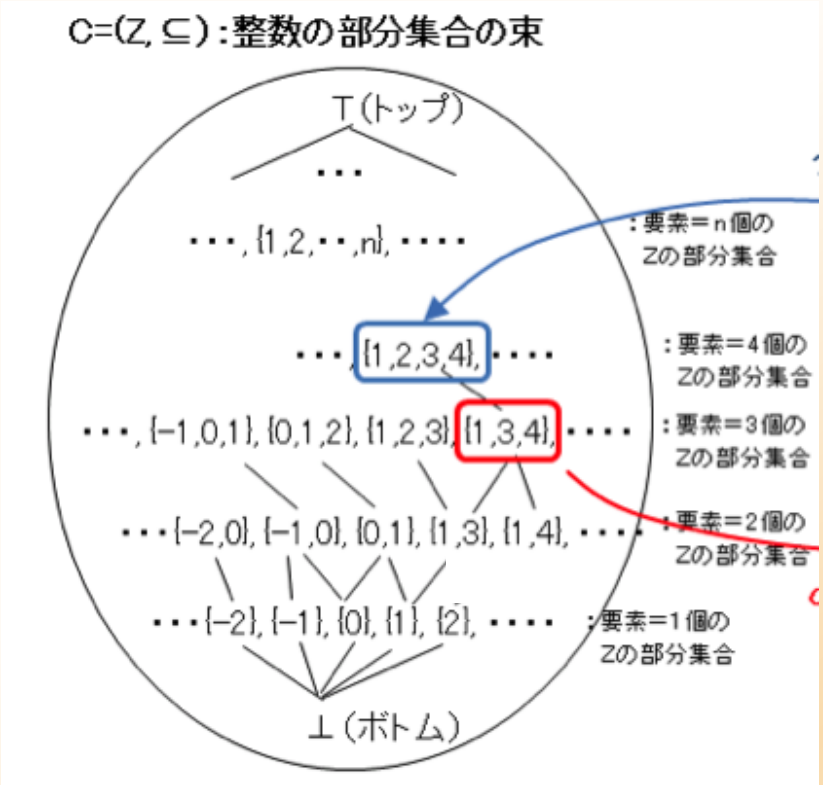
ここで、プログラムに対する抽象解釈の着眼点を実現するために「Lattice（束）」の考え方を導入します。

２．３　プログラム変数の値が作るLattice構造(1)

■プログラム変数の値の全ての部分集合を考えると、
それはLattice構造になります。
例えば、プログラムの変数xの値が整数Zの場合、
図のように、

・要素１個のZの部分集合
・要素２個のZの部分集合
・・・

⊥（ボトム）=Φ（空集合）
T（トップ）={−∞, +∞}∪Z
とすると包含関係(⊆)により
完備束になります。

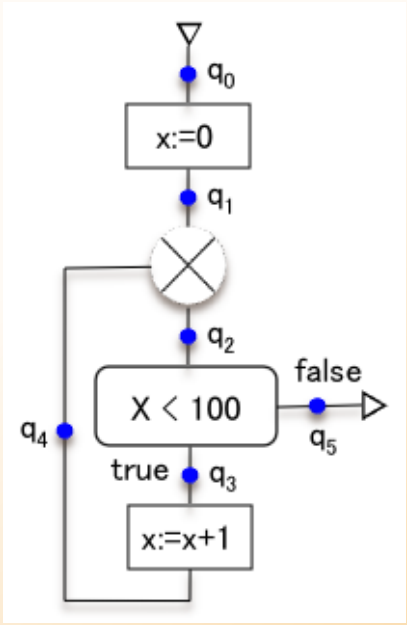


このとき、各プログラムポイント
での変数の値はLattice構造
の部分集合になります。

２．３　プログラム変数の値が作るLattice構造(2)

■各プログラムポイントで、プログラムの変数の値はプログラムの実行により変化しますが、変数の値の集合は単調増加します。

例えば、右のような１変数のプログラムを例にします。



プログラムポイント=q_2での変数xの取り得る値の集合はループ回数と共に値が増えます。

ループ1回目 = {0}
ループ2回目 = {0, 1}
ループ3回目 = {0, 1, 2}
・・・

即ち、プログラムポイント=q_2で変数xの取り得る値の集合は単調増加します。

{0}⊆{0, 1}⊆{0, 1, 2}⊆{0, 1, 2, 3}⊆・・・

２．３　プログラム変数の値が作るLattice構造(3)

言い換えると、プログラムポイント=q_2での変数xの取り得る値の集合は、

　　Zの部分集合全てが作る完備束の中の単調増加列

となります。

すると、完備束上のタルスキの不動点定理（１．５節）より

「プログラムポイント=q_2での変数xの取り得る値の集合」

の不動点が存在します。

この不動点は、プログラムが実行されてこれ以上増加しない変数xの値の集合です。

すなわち、変数xがプログラムポイントq_2で取り得る値の全てを意味します。

→従って、変数の全ての取り得る値を求めることは、増大する変数の値の集合の不動点を求めることに帰着します。

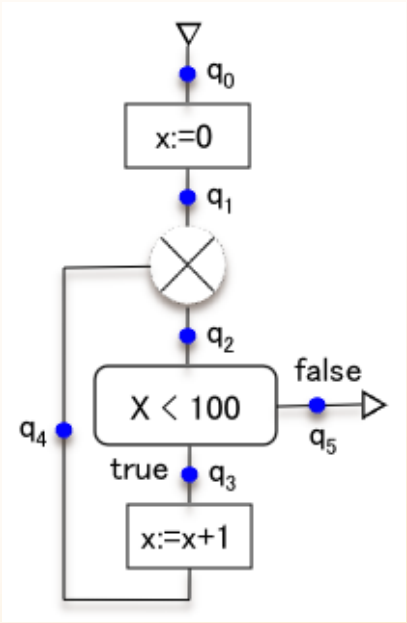
では、不動点（の近似）は具体的にどのようにして
求めるかを次に示します。
→そのためには、
①まず、プログラムの変数の値が作るLattice構造
を「ガロア接続」により扱い易い構造に変換します。

②次に、変換後の世界で不動点の近似を
「Wideningという手法」により求めます。

→次頁でこの例を示します。
尚、ここでは「プログラムのループ処理」に焦点を
当てて説明します。

2．4　ループ処理でのプログラム変数の値の変化

（例）右のような1変数のプログラムを例にします。



pt=q_2での変数xの取り得る値の集合を$S(q_2)$とし、
$S_n(q_2)$をループ処理のn回目の$S(q_2)$とすると

ループ処理の1回目：$S_1(q_2) = S_1(q_1) = \{0\}$
ループ処理の2回目：$S_2(q_2)=S_1(q_1) \cup S_1(q_4)=\{0\} \cup \{1\}=\{0, 1\}$
ループ処理の3回目：$S_3(q_2)=S_1(q_1) \cup S_2(q_4)=\{0\} \cup \{1,2\}=\{0, 1, 2\}$
・・・

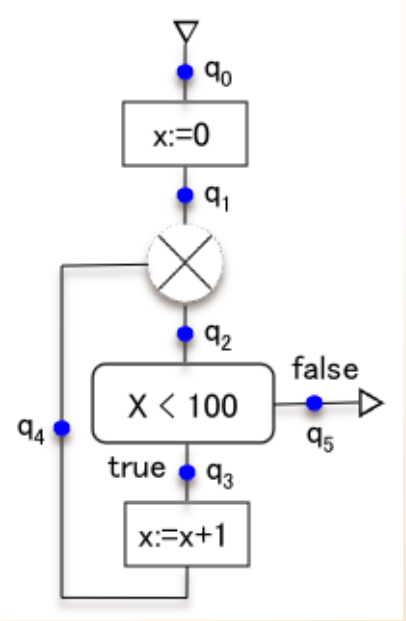
ここで、ガロア接続：$\alpha(C)=[\min(C), \max(C)]$を考えると
ループ処理の1回目：$\alpha(S_1(q_2))=[0, 0]$
ループ処理の2回目：$\alpha(S_2(q_2))=[0, 1]$
ループ処理の3回目：$\alpha(S_3(q_2))=[0, 2]$
・・・

となります。
即ち、pt=q_2で変数の値を含む区間の列は単調増加します。
$\alpha(S_1(q_2))= [0, 0] \subseteq \alpha(S_2(q_2))=[0, 1] \subseteq \alpha(S_3(q_2))= [0, 2] \subseteq \cdots$

２．５　$\alpha(S_n(q_2))$に$\alpha(S_{n+1}(q_2))$を対応させる関数F

ここで、次の関数Fを考えます；

$F(\ \alpha(S_n(q_2))\ ) = \alpha(S_{n+1}(q_2))$ ←Fは$\alpha(S_n(q_2))$にループ１回後の$\alpha(S_n(q_2))$を対応させる関数



このとき、下記が成立します；

F([0, 0])=([0, 1])

F([0, 1])=([0, 2])

F([0, 2]))=([0, 3])

・・・

従って、pt=q_2で変数の値を含む区間の列

[0, 0]⊆[0, 1]⊆[0, 2]⊆[0, 3]⊆・・・

は、Fにより次の様に表現できます；

[0, 0]⊆F([0, 0])⊆F(F([0, 0]))⊆F(F(F([0, 0])))⊆・・・

ここで、$F^n(\ \ )=F(F(\cdots(\ \ )))$と表すと、

[0, 0]⊆F([0, 0])⊆F^2([0, 0])⊆F^3([0, 0])⊆・・・

２．６　関数Fの不動点とwidening(1)

前頁から、$F^n(\ \ )=F(F(\cdots(\ \ )))$と表すと、

$[0, 0]\subseteq F([0, 0])\subseteq F^2([0, 0])\subseteq F^3([0, 0])\subseteq\cdots$　(1)

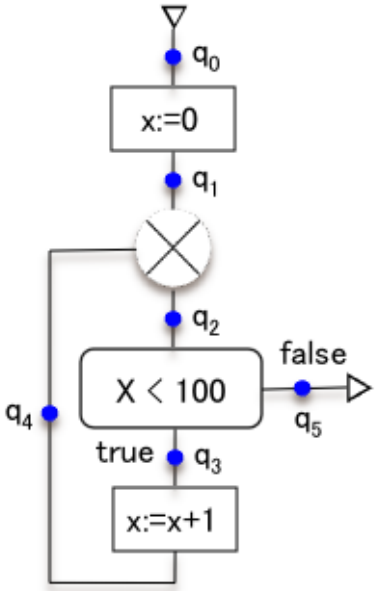
でした。

(1)式の$F^n([0, 0])$を計算すると、

$\cdots$, $F^{99}([0, 0])=[0, 99]$, $F^{100}([0, 0])=[0, 100]$

$F(F^{100}([0, 0]))=F([0, 100])=[0, 100]=F^{100}([0, 0])$

すなわち、$F^{100}([0, 0])=[0, 100]$はpt=q_2での関数Fの不動点(注)です。



(注)XがFの不動点とは、XにFを作用させても値が変らないこと。即ち、F(X)=X

プログラムの解(pt=q_5でのxの値)は、q_2での関数Fの不動点[0, 100]を

q_2の次のステップの式：x＜100

に代入することにより、区間の世界で x=[100, 100]と求まります。

今度は、ガロア接続の写像 $\gamma(I)=\{ k\in Z \mid k\in[a, b] \}$により

元の世界に戻すと、x=100が得られます。

２．６　関数Fの不動点とwidening(2)

■以上から、
プログラムのループ処理では、プログラムの変数の単調増加する値の集合の不動点を求めることがポイントとなります。

→この不動点（の近似）を簡易に求める手法としてwideningというものがあります。

■その考え方は、
「ループ処理は不動点より大きな値を持たないので、
区間を無限大にしても 不動点は変らない」
ことを利用します。
即ち、「不動点を知るためには（途中の経過は分からなくても）無限遠点でのFの値を求めれば良い」
ことが分かります。

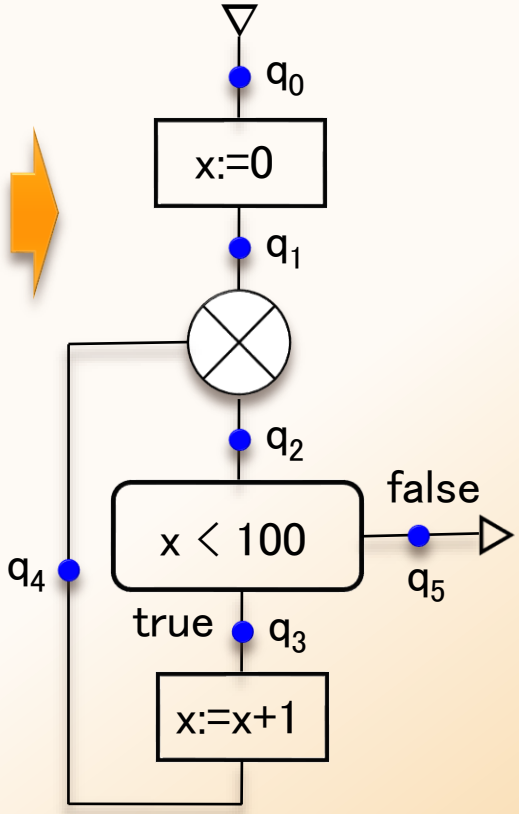
２．６　関数Fの不動点とwidening(3)

■その手順は、
　①ループの1回、2回目のxの範囲の変化を見る;
　　[0, 0]→[0, 1]に変化！
　②変化があれば区間を無限大に飛ばす:[0, ∞]
　③上記の結果[0, ∞]をFに代入する;F([0, ∞])
上記の①、②という操作は、抽象解釈では「区間のwidening」と呼ばれます;
■区間のwideningの定義（∇という記号で表します）:
　　Φ=空集合とするとき、Φ ∇I = I (I は任意の区間)
　　$[a_1, b_1] \nabla [a_2, b_2] = [\ c, d\ ]$
　　ここで、c=if $a_2 < a_1$ならば −∞, そうでないならば a_1
　　d=if $b_2 > b_1$ならば +∞, そうでないならば b_1
　（例）[0, 0]∇[0, 1]=[0, ∞]
また、③は、無限に飛ばした先から不動点まで降下するのでdecendingと呼ばれます。

2．7　抽象解釈によるプログラム解析例(1)

■これまでの説明を元に、右のプログラムの変数xの最終の値を抽象解釈で求める手順を整理します。

```
x := 0
while
      x<100 do
   x := x+1
end
```



まず、ループ処理の
①プログラムの動きを調べます;
$S_1(q_2)=\{0\}$
$S_2(q_2)=\{0, 1\}$
ガロア接続: $\alpha(C)=[\min(C), \max(C)]$を考えると、
$\alpha(S_1(q_2))=[0, 0]$
$\alpha(S_2(q_2))=[0, 1]$

②$\alpha(S_1(q_2))$と$\alpha(S_2(q_2))$のwideningを求めると、
$\alpha(S_1(q_2)) \nabla \alpha(S_2(q_2)) = [0, 0] \nabla [0, 1] = [0, \infty]$
これを$\alpha(S(q_2))_{\nabla}$と置きます。
即ち、$\alpha(S(q_2))_{\nabla} = [0, \infty]$

２．７　抽象解釈によるプログラム解析例(2)

③ $\alpha(S(q_2))_{\nabla} = [0, \infty]$ をFに代入したものが不動点となります。
Fは $\alpha(S(q_2))_{\nabla}$ にループ１回後の値を対応させる関数
なので、

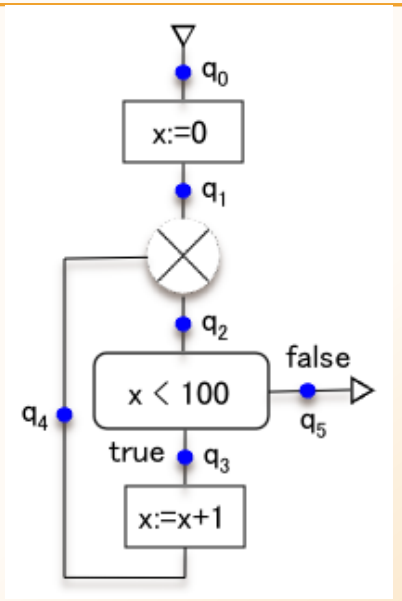
$\alpha(S(q_2))_{\nabla} = [0, \infty]$ を使ってq_3以降の処理を行います；

$\alpha(S(q_3))$ = { 条件式x<100が真となるような
$\alpha(S(q_2))_{\nabla}$の変数の区間 } = [0, 99]

$\alpha(S(q_4)) = [1, 100]$

次にループを戻って $\alpha(S(q_2))$を求めたものがF($\alpha(S(q_2))_{\nabla}$)です；

$F(\alpha(S(q_2))_{\nabla}) = (\alpha(S(q_1)) \cup \alpha(S(q_4))) = [0, 0] \cup [1, 100] = [0, 100]$



最後に $\alpha(S(q_5))$を求めると、

$\alpha(S(q_5))$ = { 条件式x<100が偽となるようなF($\alpha(S(q_2))_{\nabla}$) の変数の区間 }
=[100, 100]

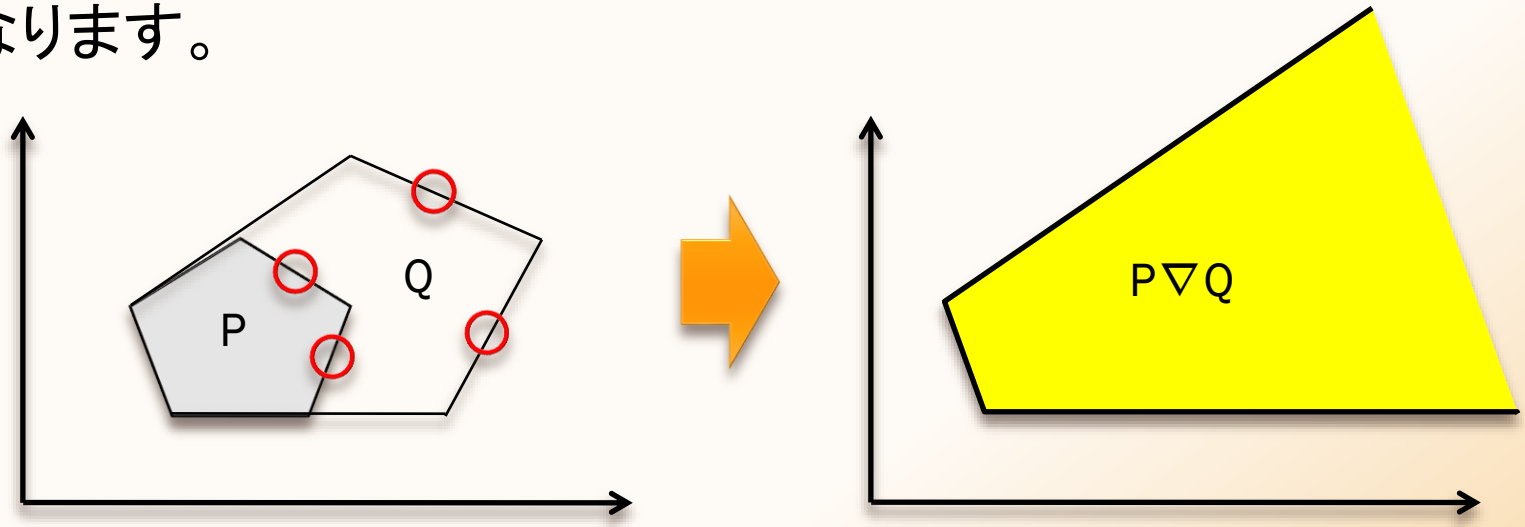
今度は、ガロア接続の写像 $\gamma(I) = \{ k \in Z \mid k \in [a, b] \}$により元の世界に戻すと、
プログラムの解 x=100 が得られます。

２．８　プログラムが2変数以上のwidening(1)

プログラムが2変数以上の場合のwideningは次の様になります。

●多面体領域のwidening

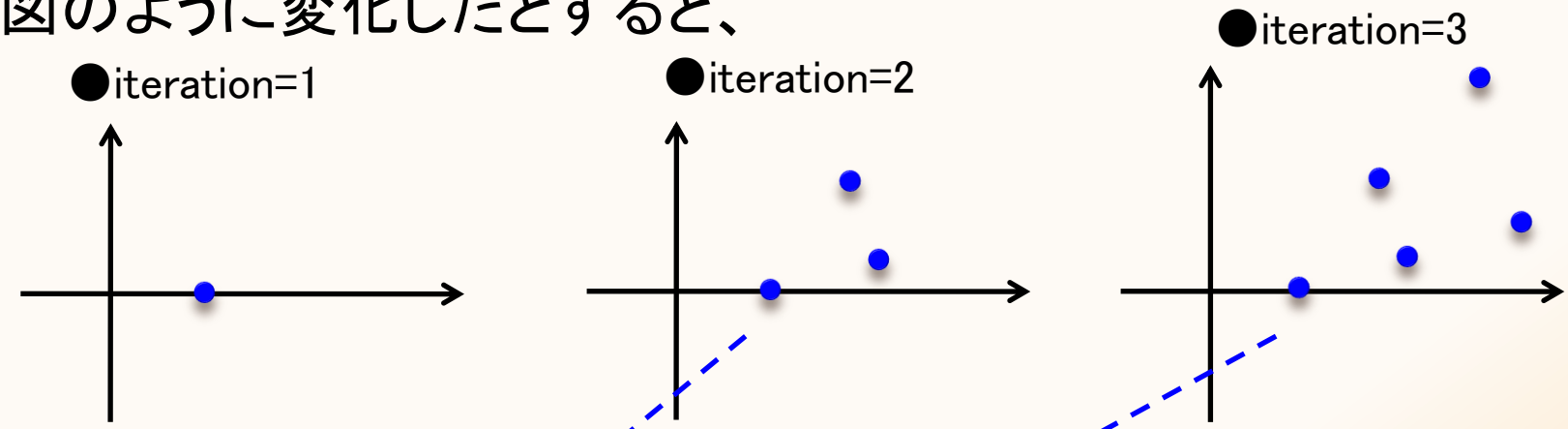
下の図のように、P, Q を P ⊑ Q となる多面体とするとき、赤丸の辺はPからQへ変るときに移動する（安定しない）線形制約です。これらの辺（線形制約）を取り除いたものがPとQのwidening P▽Q となります。



即ち、wideningとは、単調増大する多面体領域において、動きのある辺を一気に無限方向に拡げることと言うこともできます。

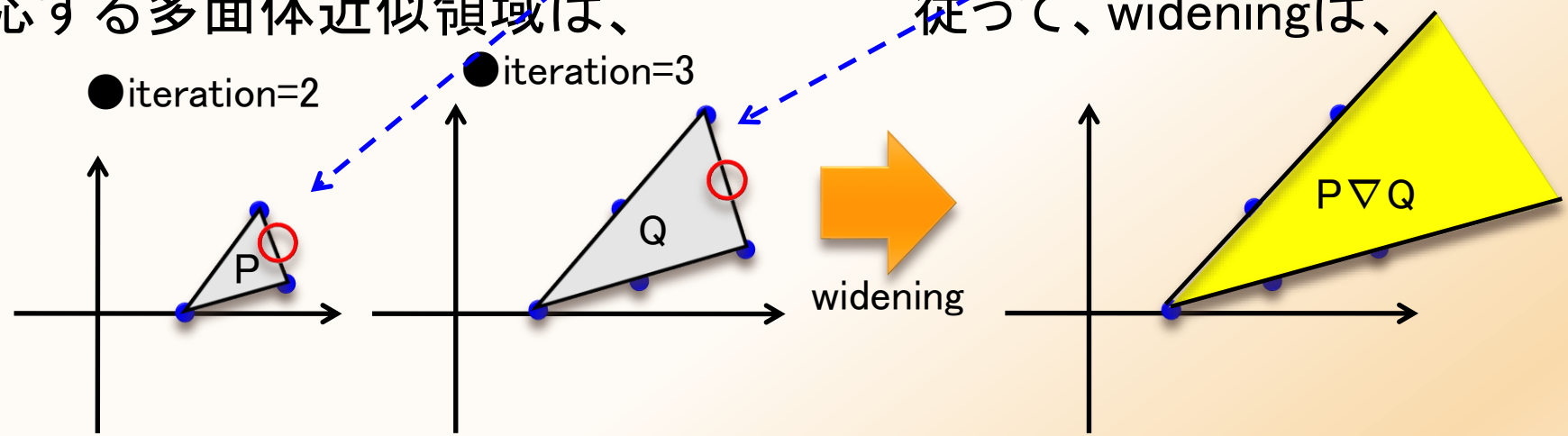
２．８　プログラムが2変数以上のwidening(2)

例えば、ループ処理のプログラムポイントで2変数（注）の値が
下図のように変化したとすると、



対応する多面体近似領域は、

従って、wideningは、



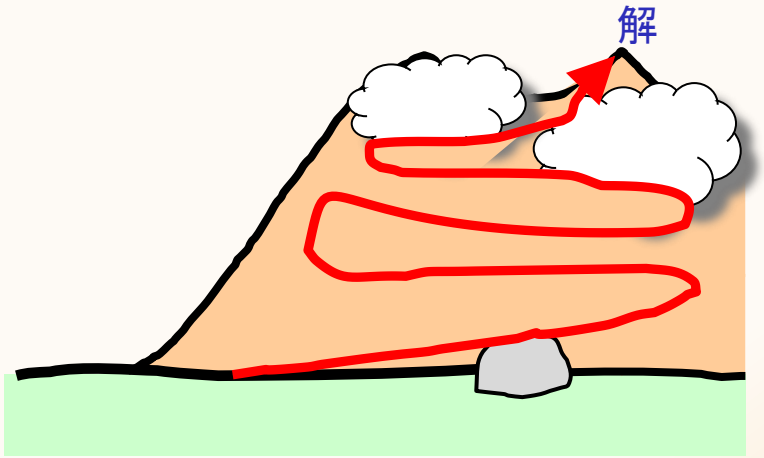
（注）ここでは、2変数の式が線形の場合を考えます

２．９　これまでの纏め

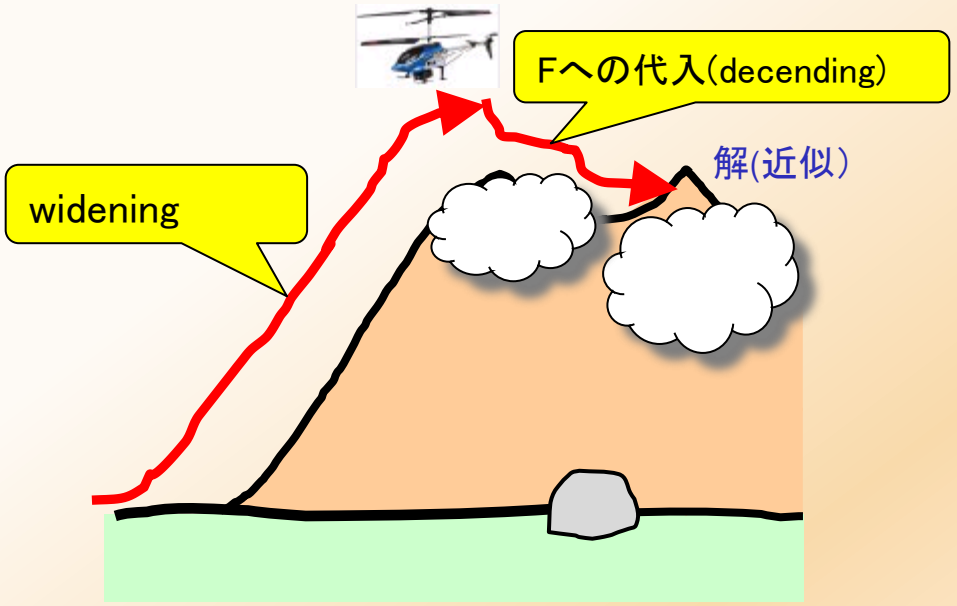
以上の例のようにwideningは、ループ処理の「単調増加が止まる（不動点）という性質を利用し、ループ処理をわずか数回計算するだけでループ処理内の変数の取り得る値（一般的には近似値）を求めることができます。

登山に喩えると・・・

●通常のループ処理は登山ルートに従って着実に登り、解に到達する。



●抽象解釈はヘリコプターで
一気に上空に上昇し(widening)、
目標付近に下降する(Fへの代入)



３．抽象解釈の方法論について

・・・最後に、抽象解釈の手法を数学のガロア理論の手法と対比することを試みます。

3. 抽象解釈の方法論について(1)

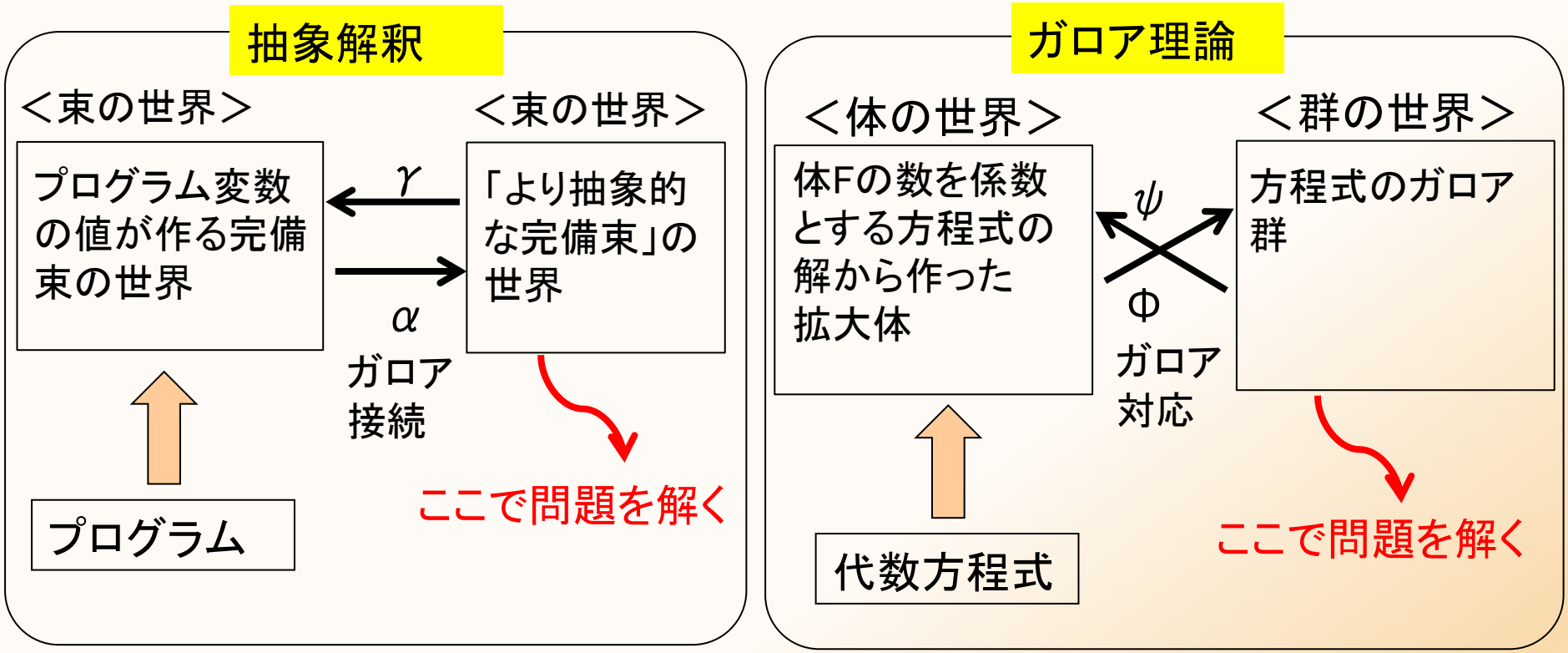
これまでの議論を振り返ると、次のようになります。

・抽象解釈は、
「プログラム変数の値の部分集合を要素とする集合」
及び
「プログラム変数の値の部分集合を含む近似領域を要素とする集合」
という2つの世界に⊆や∪、∩という演算を定義することにより、プログラムの進行による「プログラム変数の値の集合」の推移を記述します。また、2つの世界間に抽象化写像α及び具体化写像γを定義します。

・抽象解釈が求めたいものは、「プログラム変数の値の集合を含む近似領域」の不動点（変数の値の集合がこれ以上増大しない領域）です。不動点の近似値を求めるためにwidening という手法が使われます。

3. 抽象解釈の方法論について(3)

■前頁の内容を図示し、ガロア理論の方法論と対比してみます。



3. 抽象解釈の方法論について(4)

すなわち、次のように対比することができそうです。

■抽象解釈では、
①プログラムを一行一行、実際に実行することではなく、
②プログラム変数の値の集合に着目して束(Lattice)の理論に置き換え、
③さらにそれをより単純な束にガロア接続を用いて翻訳することによって
④プログラムの求めたい性質を不動点近似で特徴付けることが可能となった。

■ガロア理論では、
①方程式を代数的に解くことを、具体的に方程式を変形することではなく、
②体の拡大の理論に置き換え、
③さらに体の拡大をガロア群の言葉に(ガロア対応により)翻訳することによって、
④代数的に解くことのできる方程式の特徴付けが可能になった。
・・・現代思想 2011年4月
数学者:上野健爾氏
「ガロアの考えたこと」より

■扱う対象は異なるが共通していることは、
「対象の難しさに正攻法で正面から力ずくで攻めるのではなく、
その対象の『或る構造(注)』を抽象化し、そこで問題を解くことです。」
(注)着目する視点;
・抽象解釈の場合は、プログラムの変数の値の集合の持つ『束の構造』であり、
・ガロア理論の場合は、方程式の解の持つ『対称性の構造(体と群の構造)』です。

おわりに

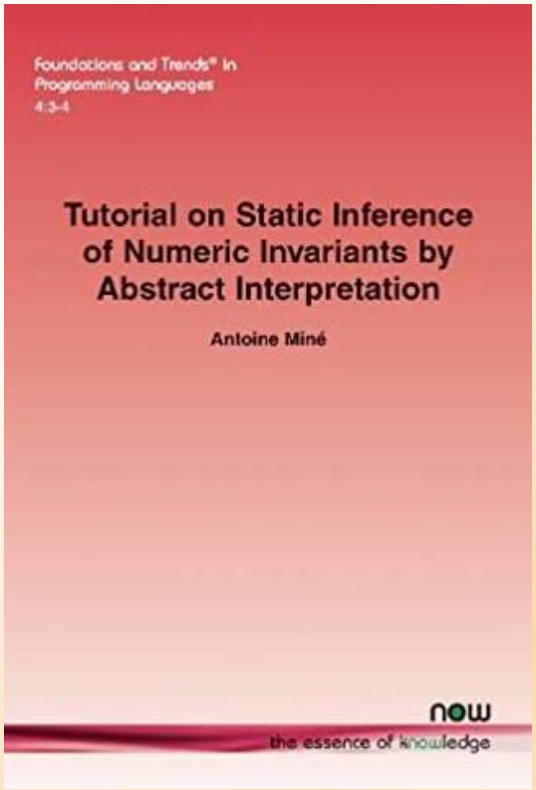
■2000年代の始め、米国の計算機科学者Joseph Goguen (1941-2006)は理論的なコンピュータサイエンスと実践的なソフトウェアのアプリケーション間に大きな断絶があり、これを理論と実践の間にある“大分水嶺(great divide)”と呼びました。そして、様々な種類の代数(主にカテゴリー論)を使用してコンピュータの使用の理論と実践間の大分水嶺を乗り越える試みをしました。彼はこれを「大分水嶺に代数の花を放る(Tossing Algebraic Flower down the Great Divide)」と表現しました。

■抽象解釈は、ソフトウェアの世界で「現代的な数学(束論等)に基づく理論」と実践を結び付けた事例の一つと言えるでしょう。

参考資料

1. P.Cousot and R.Cousot: A unified lattice for static analysis of programs by construction or approximation of fixpoints (1977)
2. P. Cousot and N. Halwachs ; AUTOMATIC DISCOVERY OF LINEAR RESTRAINTS AMONG VARIABLES OF A PROGRAM, 1978
3. Antoine Miné ; The Octagon Abstract Domain, In AST 2001 in WCRE 2001, IEEE, pages 310-319. IEEE CS Press, October 2001.
4. Flemming Nielson, Hanne Riis Nielson, Chris Hankin; Principles of Program Analysis, Springer 2005
5. Klaus Denecke, Marcel Erné, Galois connections and applications (Mathematics and Its Applications), Springer, 2004
6. Antoine Miné ; Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation, now Publishers Inc. 2017

→比較的最近出版された抽象解釈についての纏まった本(抽象解釈の教科書として最適)。



END

付録. 抽象解釈によるプログラム解析例

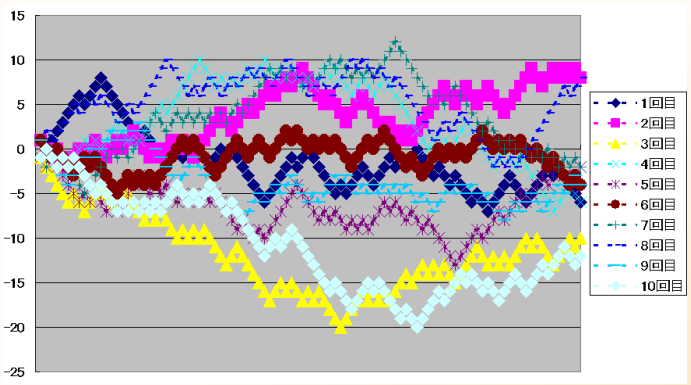
抽象解釈によるプログラム解析例1(1)

■下記はm歩の1次元ランダムウォーク(確率1/2で前後に一歩進むを決定)をM回実行するプログラムです。配列tab[a]にはaという位置に止まった回数をカウントして格納します。これを実行したときに配列tab[a]に範囲外エラー(buffer overrun)が発生しないことを抽象解釈で解いてみます。

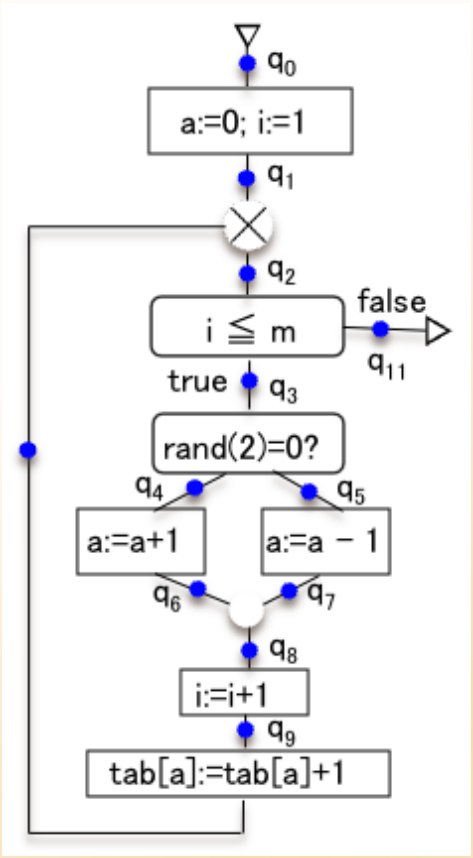
```
int tab[−m. . .m];
for i = −m to m    tab[i] = 0;
for j = 1 to M do
    int a = 0; int i=1;
    while i ≦ m do
      if rand(2) = 0
        then a = a + 1;
        else a = a − 1;
      fi
      i=i+1
    tab[a] = tab[a] + 1;
done;
```

(注)rand(n)は0からn−1までの数のうちいずれかをランダムに返す。

フローチャートでは

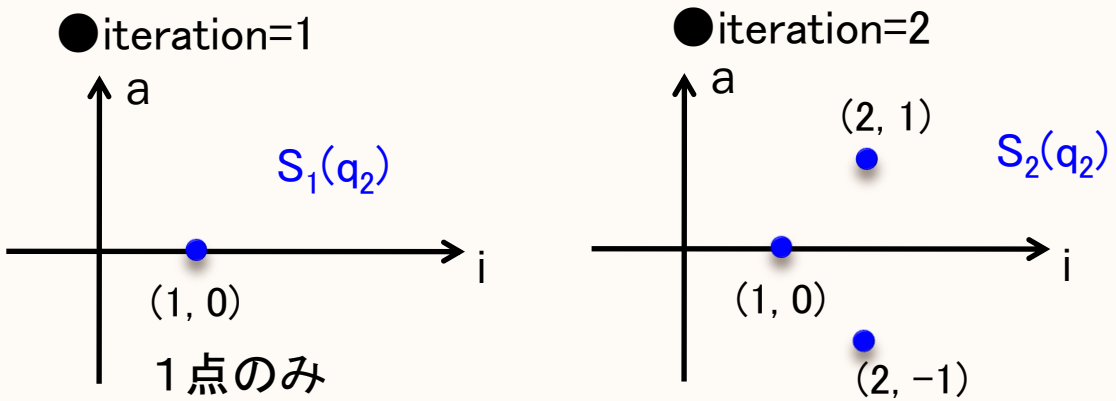


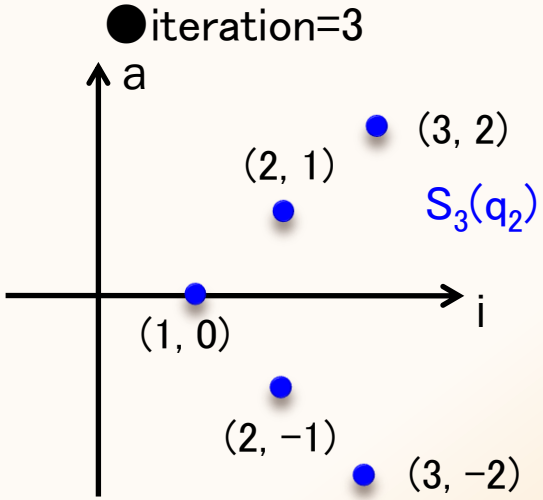
m=100歩の酔歩をM=10回試行の例

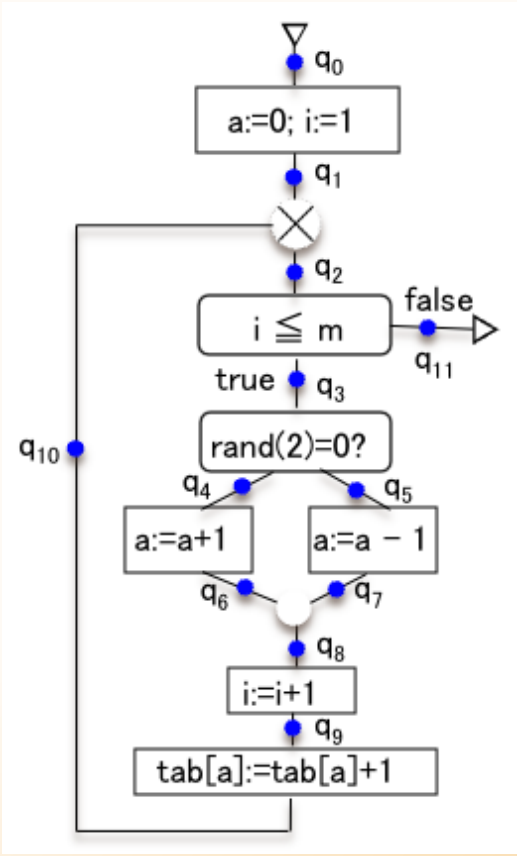


抽象解釈によるプログラム解析例1(2)

■pt=q_2での値の集合$S_1(q_2)$, $S_2(q_2)$, $S_3(q_2)$を
求めると、下図のようになります（途中の計算は省略）。

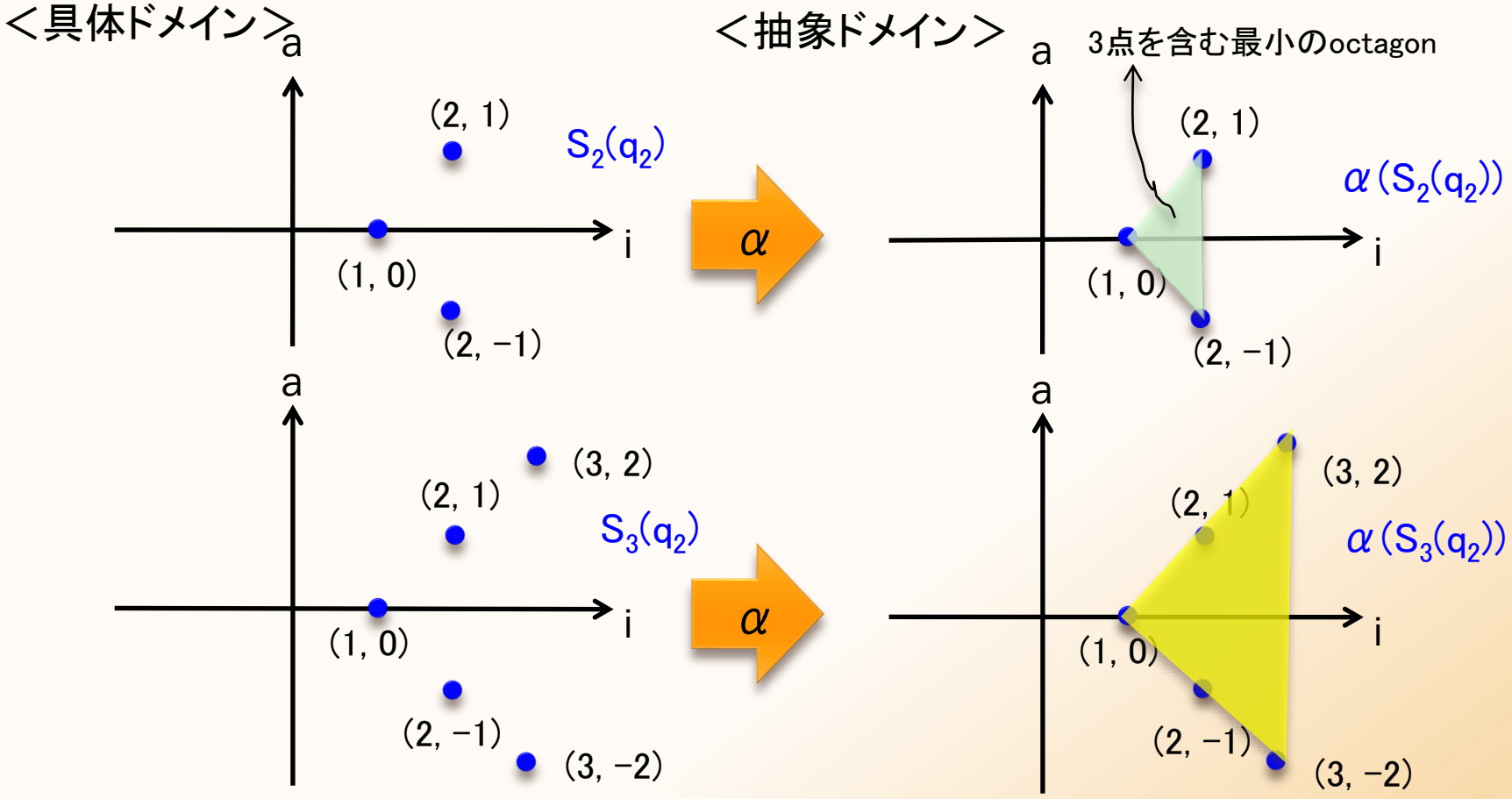






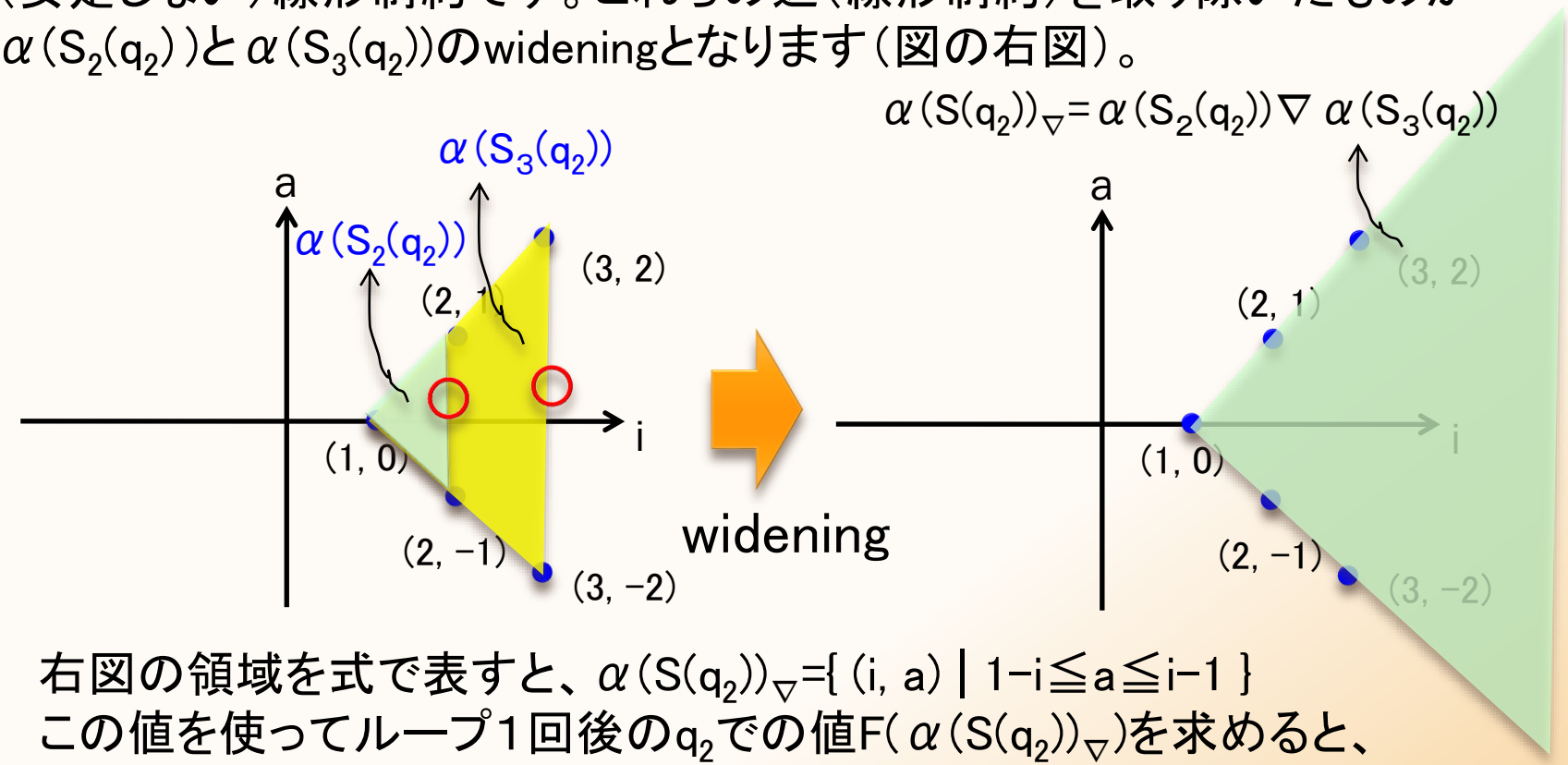
抽象解釈によるプログラム解析例1(3)

■次に、pt=q_2での具体値の集合$S_2(q_2)$, $S_3(q_2)$に対応する抽象値の集合を求めます。ここで使用する抽象領域はoctagon(八角形)とします。



抽象解釈によるプログラム解析例1(4)

■ここで、$\alpha(S_2(q_2))$と$\alpha(S_3(q_2))$のwideningを求めます；
下の左図において、赤丸の辺は$\alpha(S_2(q_2))$から$\alpha(S_3(q_2))$へ変るときに移動する（安定しない）線形制約です。これらの辺（線形制約）を取り除いたものが$\alpha(S_2(q_2))$と$\alpha(S_3(q_2))$のwideningとなります（図の右図）。



右図の領域を式で表すと、$\alpha(S(q_2))_{\nabla}=\{ (i, a) \mid 1-i \leqq a \leqq i-1 \}$
この値を使ってループ1回後のq_2での値$F(\alpha(S(q_2))_{\nabla})$を求めると、
$S(q_{11})=\{ (i, a) \mid -m \leqq a \leqq m \}$が求まり、配列tab[a]に範囲外エラーが発生しないことが分かります（途中の計算はここでは省略します）。

抽象解釈によるプログラム解析例2 (1)

■右のプログラムでzero divideが
発生しないかを抽象解釈で考えます。

pt=q3でのプログラムの値の集合$S(q_3)$は;

$$S(q_3)=\{(I, J, K) \mid I=2, J=K+5, -2^{31} \leqq K \leqq 2^{31}-1\}$$

次にループ処理のプログラムの動きを調べます;

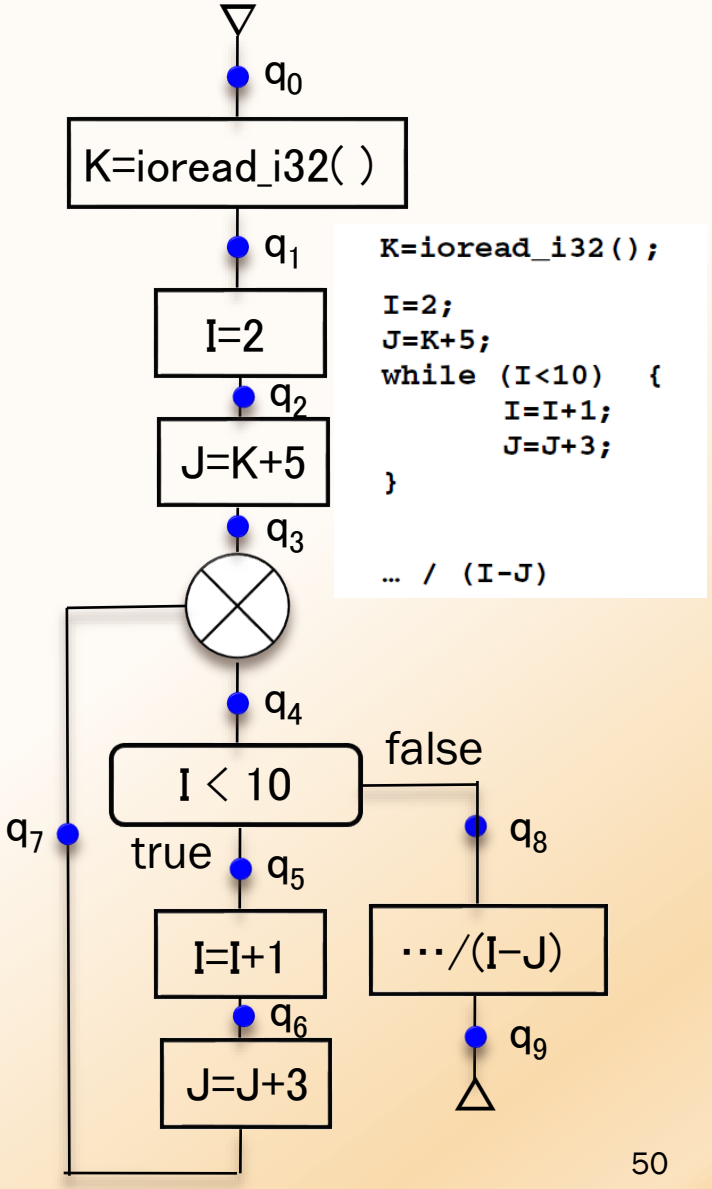
●iteration=1

$$S_1(q_4)=\{(I, J, K) \mid I=2, J=K+5, -2^{31} \leqq K \leqq 2^{31}-1\}$$

$$S_1(q_7)=\{(I, J, K) \mid I=3, J=K+5+3, -2^{31} \leqq K \leqq 2^{31}-1\}$$
$$=\{(I, J, K) \mid I=3, J=K+8, -2^{31} \leqq K \leqq 2^{31}-1\}$$

●iteration=2

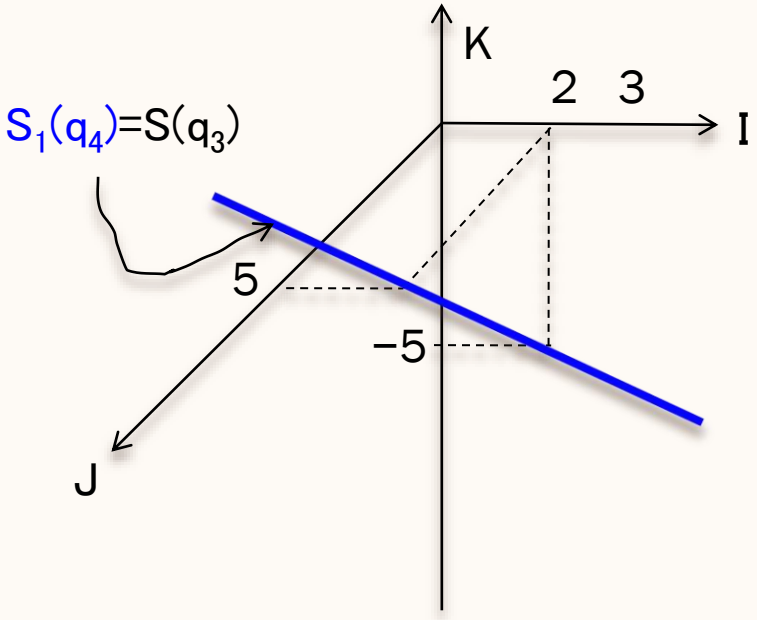
$$S_2(q_4)=S(q_3) \cup S_1(q_7)$$
$$=\{(I, J, K) \mid I=2, J=K+5, -2^{31} \leqq K \leqq 2^{31}-1\}$$
$$\cup\{(I, J, K) \mid I=3, J=K+8, -2^{31} \leqq K \leqq 2^{31}-1\}$$



抽象解釈によるプログラム解析例2 (2)

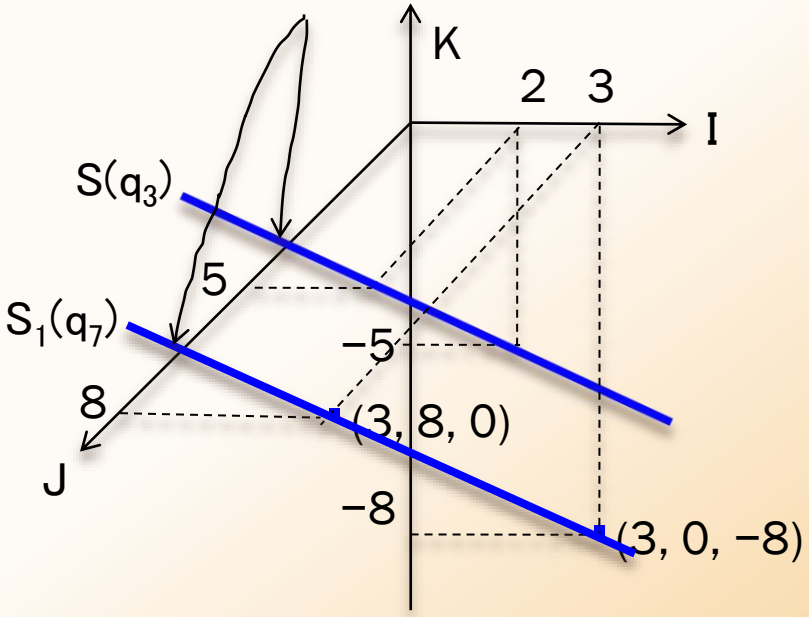
従って、pt=q_4でのプログラム値の集合$S_1(q_4)$, $S_2(q_4)$は、下図のように
3次元空間の直線になります($S_2(q_4)$は2本の平行な直線);

●iteration=1



●iteration=2

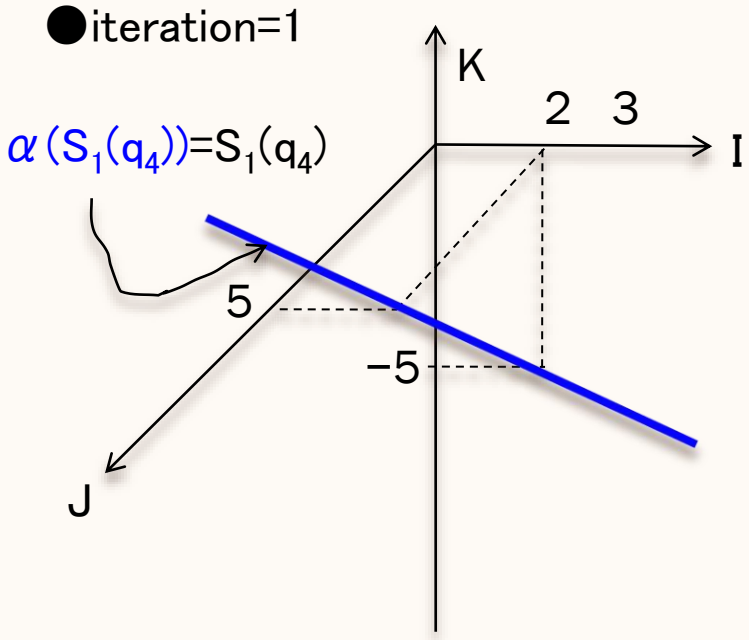
$S_2(q_4)$=S(q_3) ∪ $S_1(q_7)$

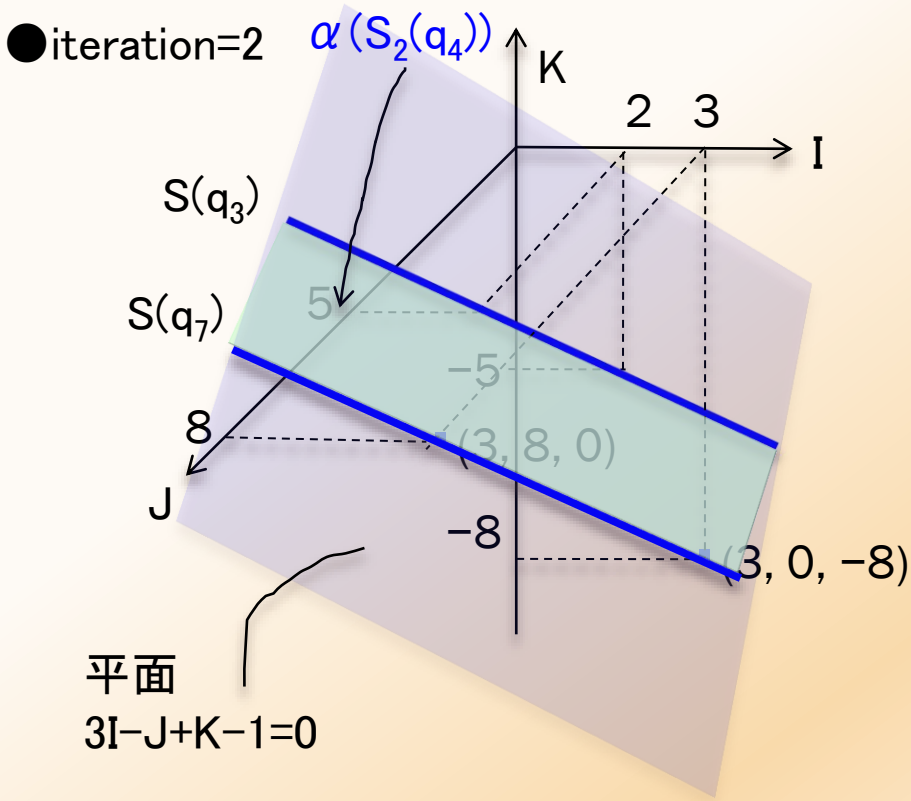


抽象解釈によるプログラム解析例2 (3)

次に、ガロア接続αによりpt=q_4での値の集合$S_1(q_4)$, $S_2(q_4)$に対応する抽象値の集合を求めます。ここで使用する抽象領域は多面体とします。

- $\alpha(S_1(q_4))$は、1本の直線なので$\alpha(S_1(q_4))=S_1(q_4)$です（下の左図）。
- $\alpha(S_2(q_4))$は、2つの直線$S(q_3)$、$S(q_7)$で囲まれた平面です（下の右図）。
  これは、平面3I−J+K−1=0の一部です（平面の式を求める計算は省略）。





抽象解釈によるプログラム解析例2(4)

■ここで、$\alpha(S_1(q_4))$と$\alpha(S_2(q_4))$のwideningを求めます；

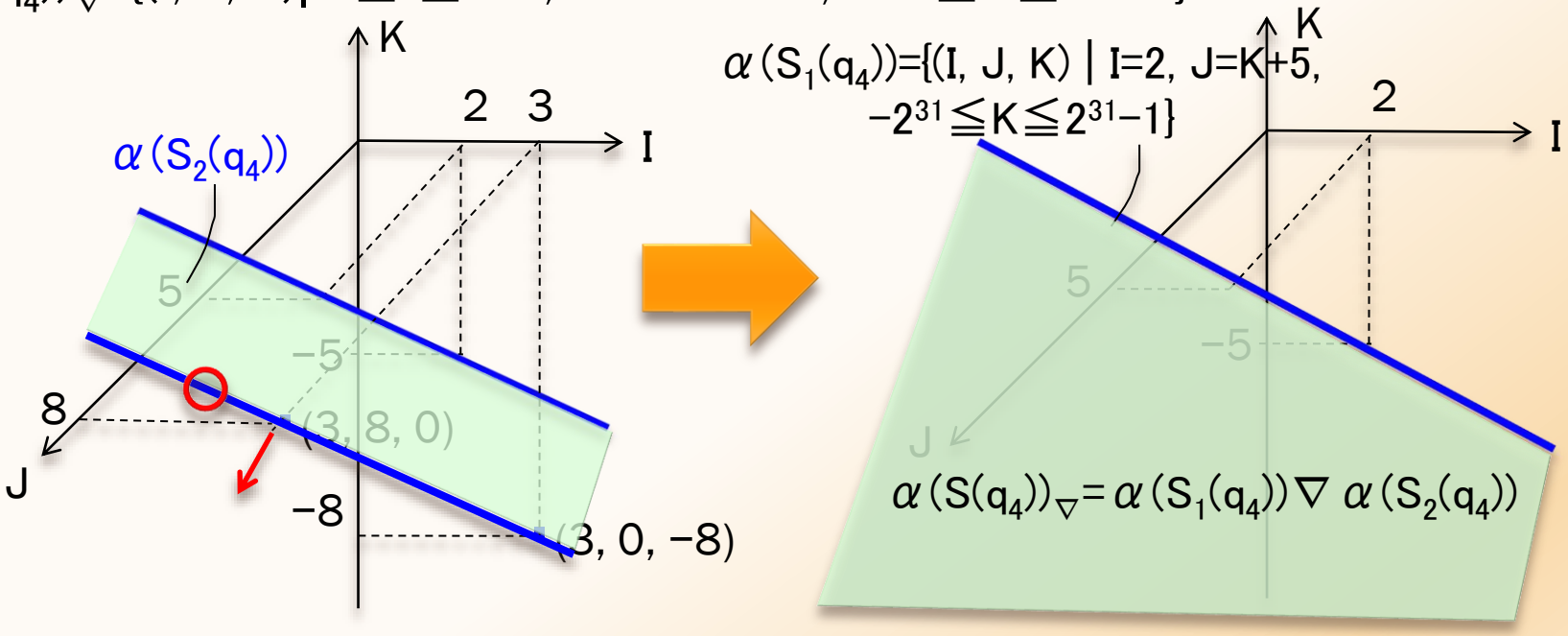
$$\alpha(S(q_4))_{\nabla} = \alpha(S_1(q_4)) \nabla\ \alpha(S_2(q_4))$$

多面体のwideningからこれは、下図のように$\alpha(S_1(q_4))$から$\alpha(S_2(q_4))$へ変るときに移動する（安定しない）辺（○の辺）を取り除いたもの（赤の→方向に半平面として無限に広がる）になります。即ち、

平面$3I-J+K-1=0$で$2 \leqq I \leqq +\infty$, $-2^{31} \leqq K \leqq 2^{31}-1$の部分。

従って、

$\alpha(S(q_4))_{\nabla} = \{(I, J, K) \mid 2 \leqq I \leqq +\infty,\ 3I-J+K-1=0,\ -2^{31} \leqq K \leqq 2^{31}-1\}$

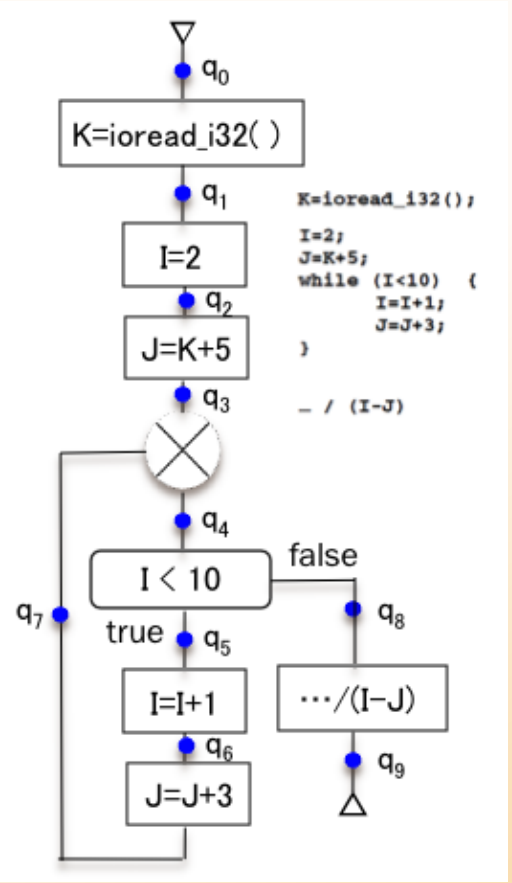


抽象解釈によるプログラム解析例2 (6)

■前頁で求めた$\alpha(S(q_4))_{\nabla}$
$=\{(I, J, K)\mid 2\leqq I\leqq+\infty, 3I-J+K-1=0, -2^{31}\leqq K\leqq 2^{31}-1\}$
を使ってdecendingを求めるためにF($\alpha(S(q_2))_{\nabla}$)を
求めます。
そのためにpt=q_4以降の処理を行います;
$\alpha(S(q_5))$
=[条件式I<10が真となるような$\alpha(S(q_4))_{\nabla}$の変数の区間]
$=\{(I, J, K)\mid 2\leqq I\leqq 9, 3I-J+K-1=0, -2^{31}\leqq K\leqq 2^{31}-1\}$

I=I+1なので
$\alpha(S(q_6))$
$=\{(I, J, K)\mid 3\leqq I\leqq 10, 3(I+1)-J+K-1=0, -2^{31}\leqq K\leqq 2^{31}-1\}$
$=\{(I, J, K)\mid 3\leqq I\leqq 10, 3I-J+K+2=0, -2^{31}\leqq K\leqq 2^{31}-1\}$

J=J+3なので
$\alpha(S(q_7))=\{(I, J, K)\mid 3\leqq I\leqq 10, 3I-(J+3)+K+2=0, -2^{31}\leqq K\leqq 2^{31}-1\}$
$=\{(I, J, K)\mid 3\leqq I\leqq 10, 3I-J+K-1=0, -2^{31}\leqq K\leqq 2^{31}-1\}$



抽象解釈によるプログラム解析例2 (7)

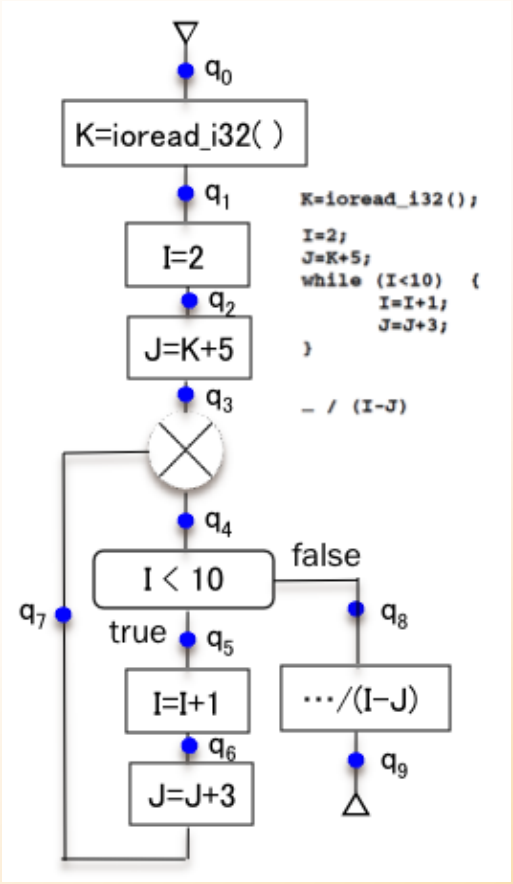
したがって、

$F(\alpha(S(q_2))_{\nabla}) = \alpha(S(q_3)) \cup \alpha(S(q_7))$

$= \{(I, J, K) \mid I=2,\ J=K+5,\ -2^{31} \leqq K \leqq 2^{31}-1\}$

$\cup \{(I, J, K) \mid 3 \leqq I \leqq 10,\ 3I-J+K-1=0,\ -2^{31} \leqq K \leqq 2^{31}-1\}$

$= \{(I, J, K) \mid 2 \leqq I \leqq 10,\ 3I-J+K-1=0,\ -2^{31} \leqq K \leqq 2^{31}-1\}$



ループ処理からは抜ける場合を求めると、

$\alpha(S(q_5))$

$= \{$条件式I<10が偽となるような$\alpha(S(q_4))$の変数の区間$\}$

$= \{(I, J, K) \mid I=10,\ 3I-J+K-1=0,\ -2^{31} \leqq K \leqq 2^{31}-1\}$

従って、

$\alpha(S(q_8)) = \{(I, J, K) \mid I=10,\ 3I-J+K-1=0,\ -2^{31} \leqq K \leqq 2^{31}-1\}$

この領域でzero divideとなるものをX_{error}とすると、

$X_{error} = \{(I, J, K) \mid (I, J, K) \in \alpha(S(q_8)),\ I=J\}$

$= \{(I, J, K) \mid I=J=10,\ K=-19\}$

すなわち、I=10, J=10, K=−19のときzero divideとなることが分かります。