計算の流れの構造に関する研究

―グラフ上の不動点問題の統一的定式化と解法―

1992年10月

玉井 哲雄

①

計算の流れの構造に関する研究

― グラフ上の不動点問題の統一的定式化と解法 ―

１９９２年 10月

玉井 哲雄

もくじ

図一覧

表一覧

はじめに

本論文は，情報の単位が流れの構造を形作ることによって計算が進むような一連の問題を，統一的に扱う試みを述べたものである．初めに，このような研究を行なうに至った個人的な経緯について，触れておきたい．

筆者は，大学院時代，伊理正夫教授のご指導のもとで，ネットワーク上の最適化問題などに関心を抱いていた．1972年，株式会社三菱総合研究所入社とともに，最初に取り組んだ仕事が，大規模な数理計画問題を取り扱う数理計画システムの開発であった．とくに，ある種の構造をもつ大規模線形計画問題に対し，その構造を活かして効率を向上させる技法を，中心的な課題とした．

続いて，大規模なネットワーク上の数理計画問題，たとえば最短路問題，最大流問題，最小費用流問題，多種流問題などを総合的に取り扱うソフトウェアの開発に従事した．ここで，硬で大規模なネットワーク上の問題を取り扱うための技術について，知見を得た．また，とくに多種流問題は，構造をもつ線形計画問題の特殊問題といえ，前の数理計画システムの開発プロジェクトとの関連も深いものであった．

さらに，1976年頃より，ソフトウェア工学の分野に取り組むこととなった．最初に手掛けたのが計算機の原始プログラムの静的な解析を行なうツールの開発であり，そのためにコンパイラの最適化手法として研究されてきたデータフロー解析技術を調べ，それを応用した．

並行して，やはりソフトウェア工学の分野で，プログラムがその仕様と合致しているか否かを機械的に検証するという，実験的なシステムの開発も試みた．そこでは，やはり原始プログラムの構造的な解析を行なうとともに，その意味についての取り扱いが必要となった．そこで，記号実行と呼ばれる技法をわれわれなりに工夫して，適用した．

これらの一連の仕事を通じて，そこに一つの共通性があることに気がついたのは，1977年頃である．大規模問題の構造の分析と分解，大規模なグラフの構造の分析と分解，最短路問題，データフロー解析，記号実行などが，グラフ上のデータの流れを扱うという点で，共通な構造をしているという認識であった．

文献を当たってみると，このような認識に基づく研究例は，すでに多く見られることが分かった．たとえば，最短路問題とデータフロー解析に関して，それぞれの問題が，実数体上の線形方程式系と類似の構造をもち，その解法と線形方程式の解法との間にある種の対応関係があることは，しばしば指摘されている（最短路に関しては例えば [8, 68]，データフローに関しては例えば [43]）．しかし，最短路問題とデータフロー解析問題とを直接結び付けて論じたものは，案外少ない．中で例外的なものとして，R. Tarjan による一連の優れた研究がある [52, 53, 55, 56]．

これらの類似性は，きわめて一般性の高い計算の構造が存在することを示唆する．その構造とは，大雑把にいえば，グラフの頂点にある性質をもったデータが存在し，頂点と頂点とを結ぶ辺の上でそのデータが変換されて，その結果が辺の終点におけるデータに結合されるというものである．すなわち，計算は，データが辺の上を流れて変換されることと，頂点で結合されること，また頂点から分配されること，によって実行される．このような構造を，計算の流れの構造と呼ぶことにする．

この計算結果は，頂点上のデータに関する不動点解になる．これを静的に捉えれば，ある頂点から各頂点に至るすべての経路を数えあげることと，ほとんどの場合等価になる．したがって，この問題は経路問題 (path problem) と呼ばれることもある [6, 55, 42]．またこれを動的に捉えれば，メッセージ交換による並列プロセスモデルとみなすことができ，すべての頂点（プロセス）が送信を行なわない定常状態が，計算結果となる．

本論文では，最短路問題，データフロー問題を含む，このような計算の流れの構造をもったグラフ上の問題を扱う．第1章で，この問題を抽象化し，グラフ上の不動点問題として統一的に定式化する．定式化には様々な方法が考えられるが，ここでは対象とする情報の構造を表すのに束を，その上での操作を表すのに束から束への関数群で一定の性質をもったものを，用いる．グラフ上の

多様な問題を束を用いて統一的に取り扱うことついては，すでに伊理による先駆的な研究がある [24, 25]．データフロー解析に束（正確には半束）の概念を最初に導入したのは G. Kildall である [32]．

第2章では，これまで研究されてきている類似の定式化との関連において，第1章で定めた定式化の特徴を議論する．

次に第3章以降で，統一的に記述された問題についての解法を考察する．線形方程式の解法に逐次型のものと消去型のものとがあるように，この問題に対しても両者の系統のアルゴリズムが考えられる．

ここではまず第3章で，逐次型の解法で一般性の高いものを提示する．このアルゴリズムは個別の問題領域では何らかの形ですでに知られているものであるが，ここで一般的に定式化した問題に対して，この方法が正当であることは，それほど自明ではない．そこでこのアルゴリズムの正当性について証明を与える．証明により，このアルゴリズムが成立するためのなるべく一般的な前提条件も明らかとなる．

さらに，この解法と個別の領域における既存の解法との関係について，効率の問題も含め議論する．

第4章で，消去型の解法を取り上げる．最短路問題やデータフロー解析で個別に工夫されてきた解法の多くは，線形方程式におけるガウス消去法などの消去型解法に対応することが知られている．しかし，これまでの消去法系統のアルゴリズム自体の記述は，まったく行列表現で表されるか（最短路問題に対する [2, 8, 68]），逆にまったくグラフ表現に従って表されるか（データフロー解析に対する [3, 5, 17, 58]）で，操作レベルでの両者の関係が明確にされていない．行列表現では，問題の疎な性質を明示的に活用できないが，一方で，グラフ表現は全体の消去順序に応じた過程が見えにくく，行列の三角分解やブロック化に対応した消去法の掃き出し戦略との対応がつけにくい．

ここでは統一的に記述された問題に対して，種々の消去型解法を系統的に導出する．その際，行列上の操作として考えられた消去法と，グラフ上の操作とを対応させる．それにより消去法アルゴリズムの構造に明確な視点を与えるとともに，個別の問題分野に帰着させた場合にどのような特性が生じるかが明らかになる．

次に第5章で，問題の構造，すなわち対象とするグラフの構造に基づいた分

解によって，消去法の効率を上げる方法を述べる．これは行列の消去順序の選択問題に対応する．初めに，とくにデータフロー解析でよく研究されている簡約可能グラフの性質を利用した既存の解法を，消去順序と LU 分解との関連という新たな視点で捉え，上三角行列に木（森）の構造を保つという性質に基づいた特徴を明確にする．次に，簡約可能グラフの性質に直接依存せず，上三角行列に極大無閉路グラフを保持するという特徴をもつアルゴリズムを提案する．このアルゴリズムは，ブロック三角化（グラフの強連結成分分解に対応）の自然な拡張になっており，簡約可能グラフの性質を利用した方法と比べ，

1. 簡約不可能なグラフに対して自然に拡張できる；
2. 特殊なデータ構造（森や 2-3 木）を保持するための複雑な処理を必要としない，

という実用的な特徴をもつ．

最後に第 6 章でまとめを行ない，また今後の研究課題として，ここで取り上げた問題の発展形態という位置づけで，並列計算システムおよび AND/OR 型のモデルについて述べる．

記法

本論文で用いる数式などの記法は，ほぼ常識的なものを用いており，必要な場合はその説明を本文中に示している．ただ，次の 2 点についてはやや特殊な記法を使っている．

1. 論理式
   次のような，論理記号を用いる．

連言	&	
選言	||	
否定	¬	
含意	→	
同値	↔	
全称記号	∀	全称論理式は ∀変数 ∈ 集合. 論理式
存在記号	∃	存在論理式は ∃変数 ∈ 集合. 論理式

とくに，連言記号と選言記号に & と || を用いるのは，∧ と ∨ を束の演算記号としてよく用いるからである．

なお，これらの論理記号を含めた演算子の順位関係については，習慣に従う．分かりにくい時は，括弧 ( ) を適宜用いる．

2. アルゴリズム

   - アルゴリズムは，入力と出力の条件，および手続きによって表す．場合によって，手続き内部で使用するデータ構造も示す．
   - 表現は，日本語に数式を混ぜたものとする．
   - 手続きの記述には，次のようなゆるい構文規則を用いる．
     - 字下げによって，処理の構造を表す．すなわち，たとえば Algol 系のプログラミング言語で **begin** と **end** で囲う一連の処理を，同じレベルの字下げで表す．
     - 代入文は，“[変数]←[式]” で表す．
     - 条件文は，“もし，[論理式] ならば，[文]” などによって表す．
     - 繰り返し文は，
       “各 [要素]∈[集合] につき，[文]（を行なう）”
       “各 [要素]∈[集合] につき，以下を行なう”
       “[論理式] の間，以下を繰り返す”
       などによって表す．
     - 注釈文は /* と */ で囲う．ただし，文の中では，（と）で囲って示すこともある．

謝辞

本論文をまとめるにあたり，終始ご指導とご教授を賜わりました，東京大学工学部計数工学科，伊理正夫教授に心より感謝申し上げます．伊理教授には学生の時より以来，長年にわたりご指導をいただいてまいりました．

すでに述べましたように，この研究のきっかけは三菱総合研究所在籍中に経験した多様な仕事から得ております．三菱総合研究所時代の上司や同僚の方々

には，大いに感謝致します．なかでも，多くのよい仕事の機会を与えていただきました反町洋一取締役，数理計画システムやネットワーク計画システムの開発に共同して取り組み，貴重な経験を共にした岡本吉晴氏，熊野長次郎氏，およびソフトウェア工学関連の仕事を通じて，多くの共同研究を行ない，筆者がデータフロー解析などの技法に接するきっかけを作られた福永光一氏（現，日本IBM基礎研究所）には，とくにお世話になりました．

この論文は，1988年に筆者が現在所属する筑波大学経営・政策科学研究科，経営システム科学専攻に移籍してから，とりまとめたものです．当専攻の同僚と学生の方々，なかでも当専攻への移籍の機会を与えて下さいました森村英典教授に，深く感謝申し上げます．

第 1 章

問題の記述

1.1 問題の定式化

1.1.1 準備

$G=(V,E)$ を有向グラフとする．ここで V は頂点の集合，E は辺の集合である．辺 e は頂点の順序対 (v,w) に対応する（順序対 (v,w) に複数の辺が対応してもよい）．この時，v を辺 e の始点，w を終点という．e から v への写像を $h(e)$，w への写像を $t(e)$ と書く．頂点 v に入る辺の集合 $\mathrm{in}(v)$ と v から出る辺の集合 $\mathrm{out}(v)$ を，それぞれ次式で定義する．

$$\mathrm{in}(v)=\{e\in E|t(e)=v\},\quad \mathrm{out}(v)=\{e\in E|h(e)=v\}.$$

また，頂点 v の先行点の集合 $\mathrm{pred}(v)$ と v の後続点の集合 $\mathrm{succ}(v)$ を，それぞれ次式で定義する．

$$\mathrm{pred}(v)=\{h(e)|e\in\mathrm{in}(v)\},\quad \mathrm{succ}(v)=\{t(e)|e\in\mathrm{out}(v)\}.$$

有向グラフ $G_0=(V_0,E_0)$ が，$V_0\subset V, E_0\subset E$ を満たす時，G_0 は G の部分グラフであるという．とくに，$E_0=E\cap(V_0\times V_0)$ であるような部分グラフを，頂点集合 V_0 によって定められる部分グラフという．

2つの頂点 v, w 間の経路とは，辺の列 $e_1, e_2, \cdots, e_k$ で，$h(e_1)=v, t(e_i)=h(e_{i+1})(i=1,\cdots,k-1), t(e_k)=w$ を満たすものをいう．とくに，v と w が同じ頂点である時，その経路を閉路という．特殊なケースとして，1つの辺で作られる閉路，すなわち頂点 v から v への辺がある場合，その辺をセルフルー

ブという．また，v から w への経路が存在する時，w は v から到達可能であるという．

閉路をもたない有向グラフを，有向無閉路グラフという．

以下では，有向グラフのみを扱うので，有向グラフを単にグラフという．また，V および E を有限集合とする有限グラフのみを考える．

L を束とする．すなわち，L は以下の性質をもつ2種類の2項演算，結び $\vee$ および交わり $\wedge$ に関して閉じている．

1. 結びも交わりもそれぞれ可換で結合的である．

$$x \wedge y = y \wedge x, \quad x \vee y = y \vee x,$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z, \quad x \vee (y \vee z) = (x \vee y) \vee z.$$

2. 次の吸収則が成り立つ．

$$x \vee (x \wedge y) = x \wedge (x \vee y) = x.$$

さらに吸収則から導かれる関係として，次の冪等則も成り立つ．

$$x \wedge x = x, \quad x \vee x = x.$$

半順序関係 $\leq$ が，次式によって定義できる．

$$x \leq y \leftrightarrow x \wedge y = x.$$

束はさらに適当な性質を仮定することにより，いくつかの種類に分類される（たとえばモジュラー束，分配束，ブール束）．以下の議論では上の基本的な性質のほかに，次のいくつかの性質を組み合わせて，場合に応じ付加的に仮定することにする．

1. 最大元と最小元の存在

   最大元1と最小元0が L に含まれる．すなわち，任意の $x \in L$ に対し，

$$1 \wedge x = x, \quad 0 \wedge x = 0.$$

2. 完備性

   L の任意の部分集合 X は，最大下界 $\inf(X)$ をもつ．すなわち，

$$\forall x \in X.\, \inf(X) \leq x \;\&\; \forall y \in L.\, ((\forall x \in X.\, y \leq x) \rightarrow y \leq \inf(X))$$

   これはまた，L の任意の部分集合 X が，最小上界 $\sup(X)$ をもつことと同等である [65]．L が完備であれば，L 自身の最小上界と最大下界は，それぞれ L の最大元と最小元になる．すなわち，上の性質1を含む．

3. 有限下降鎖 (descending chain) 条件

   L の任意の要素列 $x_1, x_2, x_3, \cdots$ が，$x_1 > x_2 > x_3 > \cdots$ を満たすならば，その列は有限である．ここで，

$$x > y \;\leftrightarrow\; y \leq x \;\&\; y \neq x.$$

   有限下降鎖条件が満たされれば，L の空でない任意の部分集合 X が，最小上界 $\sup(X)$ をもつことがいえる．とくに，L に最大元1が存在すれば，空集合を含めた任意の部分集合が最小上界をもつことになり（空集合の最小上界が1），L は完備となる．

   なお，L が有限集合の時は，有限下降鎖条件が自明に満たされる．

後の議論で，逐次型の解法を扱う場合には性質3および最大元1の存在を（したがって，性質1と2を含む），消去型の解法を扱う場合には性質2を仮定する．

F を $L \to L$ の関数の集合とする．F の要素間にも，2つの2項演算，結び $\vee$ と交わり $\wedge$ が自然に導入される．すなわち，$f, f_1, f_2 \in F$ として，

$$f = f_1 \vee f_2 \;\leftrightarrow\; \forall x \in L.\, f(x) = f_1(x) \vee f_2(x).$$

$$f = f_1 \wedge f_2 \;\leftrightarrow\; \forall x \in L.\, f(x) = f_1(x) \wedge f_2(x).$$

さらに，関数の合成演算 $\cdot$ を，次のように定義する．$f, f_1, f_2 \in F$ として，

$$f = f_1 \cdot f_2 \;\leftrightarrow\; \forall x \in L.\, f(x) = f_1(f_2(x)),$$

以降では，関数間の演算 $\vee$ と $\wedge$ のうち，主に一方のみを扱う．それを，$\wedge$ とする（この2つの演算は双対性があるので，どちらをとってもよいが，デー

タフロー解析の研究分野での習慣に従う）．そこで，F を，$\wedge$ と $\cdot$ の2つの演算が定義された代数系としてみる．

交わり $\wedge$ は可換で結合的であり，合成 $\cdot$ は結合的である．また，$\cdot$ と $\wedge$ との間に次の分配則が成り立つ．

$$(f_1 \wedge f_2) \cdot f = f_1 \cdot f \wedge f_2 \cdot f,$$

$$f \cdot (f_1 \wedge f_2) = f \cdot f_1 \wedge f \cdot f_2.$$

F は次の性質を満たすものと仮定する．

1. 演算 $\cdot$ と $\wedge$ に関して閉じている．

2. $\cdot$ の単位元として ι が，$\wedge$ の単位元として τ が，F の中に存在する．ι は恒等関数，τ は常に1を返す関数，すなわち任意の $x \in L$ につき，

$$\iota(x) = x, \quad \tau(x) = 1.$$

すなわち F は，$\wedge$ に関して可換なモノイド（単位元をもつ半群）をなし，$\cdot$ に関してやはりモノイドをなす．このような代数系を，（乗法について単位元をもつ）半環と呼ぶことがある（2.2節参照）．

なお，合成 $\cdot$ の結合性から，f^k を k 個の f を $\cdot$ で結合したものとして定義できる．f^0 は ι とする．

さらに次の性質のうちの，いずれか1つを仮定する．

3a. 単調性
任意の $f \in F$ は次の意味で単調である．

$$\forall x, y \in L.\, (x \leq y \rightarrow f(x) \leq f(y)).$$

3b. 分配性
任意の $f \in F$ は次の意味で分配的である．

$$\forall x, y \in L.\, f(x \wedge y) = f(x) \wedge f(y).$$

3c. 連続性
任意の $f \in F$ は次の意味で連続である．ただし，L の完備性を仮定する．

$$\forall X \in 2^L.\, f(\inf(X)) = \inf(\{f(x) | x \in X\}).$$

分配性は単調性を含み，連続性は分配性を含む．すなわち，この3つの性質は，下のものほど強い仮定となる．とくに，束が集合として有限集合の場合には，連続性は分配性と一致する．逐次型の解法に関しては性質3a(または3b)を，消去型の解法に関しては性質3cを仮定する．

さらに，F 上に1項演算 $*$ を導入し，次の性質を仮定することがある．

4. F は次の性質をもつ1項演算 $*$ に関して閉じている．

任意の $f \in F$ に対し，

$$f^* = \iota \wedge f \cdot f^*$$

この演算は，問題を以下のように不動点問題として定式化するためだけなら，導入する必要がない．しかし，ある種の問題に対しては，その解の表現に必要であり，また第4章で述べる消去型の解法では，本質的な役割を果たす．第4章では，$*$ 演算を上のように公理的に与えるのではなく，L の完備性と F の連続性を仮定して，基本演算から構成することを試みる．それによりこの演算の意味が明確になろう．

1.1.2 問題

問題は次のように記述できる．E に属する各辺 e に，F の要素 f_e がラベルとして結びつけられている．V に属する各頂点 v に，L の上の変数 x_v がラベルとして結びつけられていて，次の基本方程式を満たす．

$$x_v = \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}) \tag{1.1}$$

ここで $\wedge$ は可換で結合的なので，右辺のような拡張記法が可能である．問題は，この方程式の解として x_v の値を求めることである．

(1.1) の解は一般に不定となりうるので，それを避けるために次の形で考えることがある．各頂点 v に対し定数 $b_v \in L$ を与えて，

$$x_v = (\bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)})) \wedge b_v \tag{1.2}$$

多くの場合は，ある特定の頂点 s に対してのみ $b_s \neq 1$ で，他の頂点 v に対しては $b_v = 1$ である．このようなとき，s を出発点と呼ぶ．この場合の方程式を陽に書けば，次のようになる．

$$\begin{aligned} x_s &= (\bigwedge_{e \in \mathrm{in}(s)} f_e(x_{h(e)})) \wedge b_s, \\ x_v &= \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}) \quad (v \neq s). \end{aligned} \tag{1.3}$$

とくに，$\mathrm{in}(s) = \emptyset$ の場合は，次のようになる．

$$\begin{aligned} x_s &= b_s, \\ x_v &= \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}) \quad (v \neq s). \end{aligned} \tag{1.4}$$

これらの定式化の間の関係は，次のようである．

(1.1) において，ある頂点 s から他の任意の頂点 v に到達可能なとき，後に述べるような解法により，x_v を x_s の関数として解くことができる．とくに x_s 自身については，$x_s = f(x_s)$ という形の不動点方程式になる．この x_s に関する不動点方程式の一つの解を b_s として，x_s を b_s に固定すれば，元の方程式 (1.1) は，x_s に関する式を $x_s = b_s$ にかえることにより，(1.4) に帰着する．

一方，x_s に関する式を削除し，辺 $e \in \mathrm{out}(s)$ の終点 $v = t(e)$ については，$b_v = f_e(b_s)$ を与え，その他の $v(\neq s)$ については $b_v = 1$ を与えれば，(1.2) に帰着する．

また，(1.2) で，V と別に新たに頂点 s と変数 x_s を導入し，s から $v \in V$ にそれぞれ辺 e_{sv} をもうけて，そのラベル $f_{e_{sv}}$ を $f_{e_{sv}}(x) = b_v$ と定め，(1.2) の右辺からは b_v の項を除くと，出発点 s のみに b_s を与えた (1.4) に帰着する．ただし，$b_v = 1$ であるような頂点 v には s からの辺を作る必要はない．なお，b_s は任意でよいから，これを $x_s = x_s$ というような不定な形にしておけば，(1.1) に帰着するともいえる．

まったく同様に，(1.3) は，1 点を別にもうけて s との間に同じような辺を作れば，ただちに (1.4)（または (1.1)）になる．

1.1.3 解の意味

方程式の不動点解の意味を，より直観的に明らかにするために，次のように経路に沿って計算された束の値の交わりという概念を導入する．

頂点 v から頂点 w へのあらゆる経路の集合を，$\mathrm{path}(v, w)$ とする．経路 $p \in \mathrm{path}(v, w)$ の辺の列を $e_1, e_2, \ldots, e_k$ とした時，p によって定まる関数 $f_p \in F$ を，次式で定義する．

$$f_p = f_{e_k} \cdots f_{e_2} \cdot f_{e_1}$$

ただし，経路が空列の時は $f_p = \iota$ とする．

頂点 v に対して，出発点 s からの経路全体についての交わりをとった解 (meet over all paths (通称 MOP) solutions; 以下ではこれを経路解と呼ぶ) とは，次の形式のものである．

$$x_v = \bigwedge_{p \in \mathrm{path}(s,v)} f_p(b_s) \tag{1.5}$$

ただし，L は完備であると仮定する．その時，上式は $\mathrm{path}(s, v)$ が無限集合でも意味をもつ．

具体的な問題の多くでは，(1.5) の形の解が実際的な意味をもつことが多い．この経路解と不動点解との関係については，次の定理が成り立つ．

定理 1.1 *F の要素が単調性を満たす時，式 (1.4) の解は，次の性質を満たす．*

$$x_v \leq \bigwedge_{p \in \mathrm{path}(s,v)} f_p(b_s) \tag{1.6}$$

この定理は，次の補題からただちに導かれる．

補題 1.1 *F の要素が単調性をもつとき，方程式 (1.4) の不動点解 x_v は，出発点 s から v に至る任意の経路 p に対して，次の不等式を満たす．*

$$x_v \leq f_p(b_s)$$

証明 $p=(e_1,e_2,\ldots,e_k)$ とし，対応する頂点列を $v_0(=s),v_1,v_2,\ldots,v_k(=v)$ とする．また，p の先頭から e_i までの部分経路（すなわち部分列）を p_i と書く．このとき，次の式が成り立つことを，帰納法で示す．

$$x_{v_i} \leq f_{p_i}(b_s),\ (i=0,\ldots,k) \tag{1.7}$$

$i=0$ の場合，$x_{v_0}=b_s$ であり，$f_{p_0}=\iota$ であるから，自明に成り立つ．

次に，$i-1$ までは (1.7) が成立しているとして，i に対して (1.7) が成り立つことは次のように示される．

$$\begin{aligned} x_{v_i} &= \bigwedge_{e\in \mathrm{in}(v_i)} f_e(x_{h(e)}) \\ &\leq f_{e_i}(x_{v_{i-1}}) \\ &\leq f_{e_i}(f_{p_{i-1}}(b_s)) \\ &= f_{p_i}(b_s) \end{aligned}$$

式 (1.7) で，とくに $i=k$ とすれば補題の不等式に帰着する．（証明終り）

さらに，F が分配性を満たす時，式 (1.6) の両辺が等しいような不動点解が存在すること，すなわち経路解が最大の不動点解と一致する ([32]) ことがいえるが，それについては，第3章で述べる．

1.2 問題例

いくつかの具体的な問題が，上記の定式化によって扱えることを示す．

1.2.1 到達可能性

グラフ上の1つの頂点から到達可能な頂点の集合を求める問題である．

出発点 s から到達可能な頂点を求めるものとする．L として $(0,1)$ のブール束をとり，すべての $e\in E$ に対し f_e を ι（恒等関数）とする．$b_s=0$ として方程式 (1.4) の極大な不動点解を求め，解 x_v が0なら s から v に到達可能，1なら s から v に到達不可能と解釈する．

[例]

図1.1のグラフを考える．出発点を頂点1として，方程式 (1.4) を書き下すと次

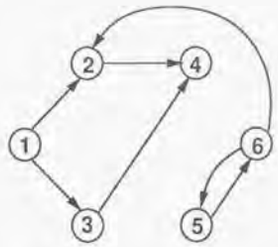


図 1.1 到達可能性問題の例

のようになる．

$$\begin{aligned} x_1 &= 0 \\ x_2 &= x_1 \wedge x_6 \\ x_3 &= x_1 \\ x_4 &= x_2 \wedge x_3 \\ x_5 &= x_6 \\ x_6 &= x_5 \end{aligned} \tag{1.8}$$

この時，極大な不動点解と経路解は一致し，

$$\begin{aligned} x_1 &= x_2 = x_3 = x_4 = 0, \\ x_5 &= x_6 = 1. \end{aligned}$$

となる．すなわち，頂点1から2, 3, 4へは到達可能だが，5, 6へは到達不可能という結果が得られる．ただし，不動点解として極大であるという条件をつけないと，x_5 と x_6 も0とするものも解となってしまい，経路解と一致しない．

なお，よく知られるように，グラフの連結成分，強連結成分，極大木などを求める問題，また行列のブロック分解，ブロック三角分解などが，すべてこの到達可能性の問題と関連する．

1.2.2 最短路問題

グラフの各辺に“距離（または費用）”が数値として与えられて，2つの頂点 v, w 間を結ぶ経路の最短距離を求める問題である．ここで経路の距離とは，経路上の辺に与えられている距離の合計をいう．最短距離を与える経路を最短路という．通常は，最短距離のみでなく最短路を求めることも要求される．なお，用いられるアルゴリズムの違いから，1点から他のすべての頂点への最短路を求める問題と，全頂点間の最短路を求める問題とに区分されることが多い．

1点から他点への問題は，次のように定式化される．出発点を s とする．L として非負整数が通常の順序関係によってつくる束をとる．最大元として，∞ を L に加える．$\wedge$ に2数の小さい方をとる演算 min を対応づける（双対的に $\vee$ は max となるが，実際には必要としない）．

各辺 e に対応づけられた距離 d_e により，関数 f_e は

$$f_e(x) = x + d_e,$$

となる．$b_s = 0$ として方程式 (1.4) を解くと，解 x_v は出発点 s から v への最短距離となる．また，最短路は，3.6.3 節に述べるような方法で求められる．

全頂点間の最短距離を求める問題は，1点から他点への問題を繰り返し解くという形で考えてもよいが，一挙に求める形に定式化し解く方法も知られている．これについては，消去型の解法のところで述べる．

なお，ここでは，距離を非負整数と仮定したが，これは L が有限下降鎖条件を満たすようにするために，おいた仮定である．完備性の条件を仮定するのであれば，非負実数に無限大 ∞ を加えたものでもよく，さらに全実数に無限大 ∞ と無限小 $-\infty$ を加えたものでもよい．有限下降鎖条件は，逐次型の解法の停止性のために仮定するものであるが，後で述べるように，最短路問題に関しては，もっと緩い仮定のもとでも逐次解法が有限回で収束する．

[例]

図 1.2 に簡単な最短路問題の例を示す．辺に記した数値が距離を表す．

出発点を頂点 1 として，$\wedge \Rightarrow \min$, $f_e(x) = x + d_e$ の対応で方程式 (1.4) を書き下すと次のようになる．

$$\begin{aligned} x_1 &= 0 \end{aligned}$$

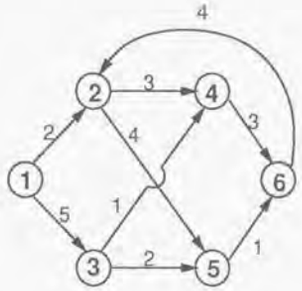


図 1.2 最短路問題の例

$$\begin{aligned} x_2 &= \min(x_1 + 2, x_6 + 4) \\ x_3 &= x_1 + 5 \\ x_4 &= \min(x_2 + 3, x_3 + 1) \\ x_5 &= \min(x_2 + 4, x_3 + 2) \\ x_6 &= \min(x_4 + 3, x_5 + 1) \end{aligned} \tag{1.9}$$

この不動点解は経路解と一致する．たとえば，x_6 の解は 7 となるが，これは頂点列 $(1,2,5,6)$ に対応する経路上の距離であり，他の経路（$(1,2,4,6), (1,3,4,6), (1,3,5,6)$ など）の距離より短い．

1.2.3 ネットワークの信頼性問題

ネットワークの故障に対する信頼性を求める問題である [46]．グラフの各辺 e に，ある時点でそれが故障により切断されていることがない（すなわち接続している）という確率 p_e が与えられているとする．その時，頂点 v, w 間が接続されている確率という意味での信頼性を求める．

ここで，各辺に起こる故障の間の独立性が問題となる．そこで理論上は，互いに独立な故障要因 $A_i (i = 1, \ldots, k)$ が数え挙げられるとし，各要因 A_i の生起確率を Q_i として，辺が接続されている確率 p_e は $P_i = 1 - Q_i (i = 1, \ldots, k)$

の多項式で表現されるとする．各辺に故障が起こるという事象が互いに独立なら，各 p_e をそのまま P_i と考えればよい．

束 L として，P_i から次の演算によって生成されるすべての多項式をとる．

1. P_i および 0 と 1 は，L の要素．

2. 項 $P_{i_1}P_{i_2}\cdots P_{i_n}$ は，L の要素．ただし，$a \neq b$ ならば，$i_a \neq i_b$．項 p に含まれる変数の集合を var-set(p) と書く．

   2 つの項 p, q から項を作る次の演算が定義される．

$$p \wedge q \equiv \prod_{P_i \in \text{var-set}(p) \cup \text{var-set}(q)} P_i$$

   なお，

$$p \wedge 1 = 1 \wedge p = p,$$

$$p \wedge 0 = 0 \wedge p = 0.$$

3. 2 つの項 p, q から次の演算で作られる式は，L の要素．

$$p \vee q \equiv p + q - (p \wedge q)$$

   なお，

$$p \vee 1 = 1 \vee p = 1,$$

$$p \vee 0 = 0 \vee p = p.$$

4. 演算 $\wedge$ と $\vee$ は，分配則を用いて式同士の演算に拡張される．また，多項式の性質から，どちらの演算も結合則と交換則が成り立つ．

辺 e の接続確率 p_e としてこのように定められた束 L の要素，すなわち P_i の多項式を割り当て，ラベルとなる関数 f_e は，

$$f_e(x) \equiv p_e \wedge x,$$

で与える．なお，ここでは習慣に合わせて，積に相当する演算を $\wedge$ で，和に相当する演算を $\vee$ で定義したので，基本方程式の $\wedge$ は，この問題に関しては $\vee$ に読み替える必要がある．

[例]

図 1.3 にネットワーク信頼性問題の例を示す．ここでは各辺に起こる故障は互いに独立なものとし，p_e を要素とする多項式で考える．

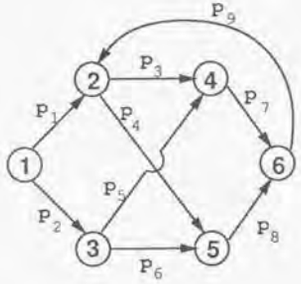


図 1.3 ネットワーク信頼性問題の例

出発点を頂点 1 として，上記の対応で方程式 (1.4) を書き下すと次のようになる．

$$\begin{aligned}
x_1 &= 1 \\
x_2 &= (p_1 \wedge x_1) \vee (p_9 \wedge x_6) \\
x_3 &= p_2 \wedge x_1 \\
x_4 &= (p_3 \wedge x_2) \vee (p_5 \wedge x_3) \\
x_5 &= (p_4 \wedge x_2) \vee (p_6 \wedge x_3) \\
x_6 &= (p_7 \wedge x_4) \vee (p_8 \wedge x_5)
\end{aligned} \tag{1.10}$$

この解として，たとえば x_6 は次のようになる．

$$x_6 = (p_1 \wedge p_3 \wedge p_7) \vee (p_1 \wedge p_4 \wedge p_8) \vee (p_2 \wedge p_5 \wedge p_7) \vee (p_2 \wedge p_6 \wedge p_8)$$

この式の各項は，頂点 1 から 6 に至る閉路を含まない経路に対応している．こ

れを多項式として展開すると，次のようなかなり長い式になる．

$$\begin{aligned} x_6 = \; & p_1p_3p_7 + p_1p_4p_8 + p_2p_5p_7 + p_2p_6p_8 - p_1p_2p_3p_5p_7 - p_1p_2p_4p_6p_8 - \\ & p_1p_3p_4p_7p_8 - p_2p_5p_6p_7p_8 - p_1p_2p_3p_6p_7p_8 - p_1p_2p_4p_5p_7p_8 + \\ & p_1p_2p_3p_4p_5p_7p_8 + p_1p_2p_3p_4p_6p_7p_8 + p_1p_2p_3p_5p_6p_7p_8 + \\ & p_1p_2p_4p_5p_6p_7p_8 - p_1p_2p_3p_4p_5p_6p_7p_8 \end{aligned}$$

1.2.4 有限オートマトン

Kleene によって最初に研究された，有限オートマトンとそれに受理される正規言語の関係を解析する問題である [2].

有限オートマトンの状態にグラフの頂点が対応し，遷移に辺が対応する．頂点の 1 つが開始状態に対応し，また最終状態に対応する頂点が 1 つ以上ある．辺はその始点で受理される文字（アルファベットと呼ばれる有限集合 Σ の要素）によって，ラベルづけられている．

アルファベット Σ 上の正規表現を考える．正規表現は，2 種類の 2 項演算 ($\cdot$, $\cup$) と 1 つの 1 項演算 $*$ で生成される．各正規表現は，Σ の要素を文字とした文字列の集合の部分集合に写像される．そこで，束 L として，この文字列集合の部分集合のクラスが作る集合束をとる．$\wedge$ は集合の和に対応し，正規表現では $\cup$ に対応する．

辺 e のラベルとなる f_e は，辺 e 上の文字を a_e として，$f_e(x) = x \cdot a_e$ により定義される．ここで，演算 $\cdot$ は正規表現の演算であるが，結果を文字列集合のクラスに写像したものとして解釈している．その意味は，文字列集合 x の各要素の末尾に文字 a_e を連接してえた文字列の集合となる．

関数空間 F の上の 1 項演算として導入した $*$ が，正規表現における $*$ に対応する．

問題 (1.3) の出発点を開始状態に対応させ，初期値 b_s として空列からなる集合を与えた時，最終状態に対応する頂点の解が，この有限オートマトンで受理される言語を与えることになる．

[例]

図 1.4 に有限オートマトンの例を示す．アルファベット $\Sigma = \{a, b, c\}$ として，各辺に受理される文字を示している．

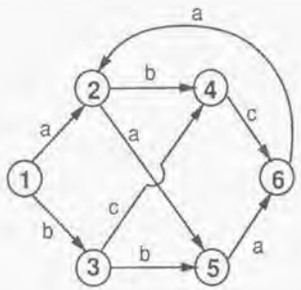


図 1.4 有限オートマトンの例

開始状態を頂点 1 として，方程式 (1.3) を書き下すと次のようになる．

$$\begin{aligned} x_1 &= \epsilon \\ x_2 &= x_1 \cdot a \cup x_6 \cdot a \\ x_3 &= x_1 \cdot b \\ x_4 &= x_2 \cdot b \cup x_3 \cdot c \\ x_5 &= x_2 \cdot a \cup x_3 \cdot b \\ x_6 &= x_4 \cdot c \cup x_5 \cdot a \end{aligned} \tag{1.11}$$

終了状態を頂点 6 とすると，解 x_6 はこの有限オートマトンで受理される言語を表す．その解は，次のようになる．

$$x_6 = (abc \cup aaa \cup bcc \cup bba)(a(bc \cup aa))^*$$

ただし表現としては正規表現を用いている（束として見る時は，文字列集合のクラスに写像して考える必要がある）．演算記号 $\cdot$ は省略した．

1.2.5 データフロー解析

データフロー解析は，手続き的なプログラムにおける実行の流れを表現したフローグラフ上で，到達する定義，使える式，忙しい式，活きている変数，な

どを解析する問題の総称である．Kildall [32] により，半束を用いてこれらを統一的に扱う方法が示された．

フローグラフは通常，頂点としてプログラムの実行単位をとり，辺によって実行の順序関係を表すように，定義される．Kildall を初めとして，データフロー解析を扱ったほとんどの論文では，フローグラフをこのように定めている．われわれの定式化では，実行単位を辺に対応させ，辺と頂点との接続関係によって，実行の流れが分岐したり合流したりする関係を表すようにする．それにより，データの状態が各頂点で定義され，その状態を次の状態に変換する関数が辺の上で定義されて，状態遷移の流れがグラフの構造に自然に対応する．従来のフローグラフの定義の仕方では，データの状態を各頂点の前と後でそれぞれ別に考える必要があった．

なお，実行単位としては，たとえば次のようなものが考えられる．

1. プログラムをコンパイラで翻訳する際に，中間的に生成されるコードの各命令（いわゆる 3 つ組や 4 つ組）
2. 手続き的なプログラムの各実行文
3. モジュール（プロシージャ，サブルーチン，など）

L として，対象とするデータの性質（たとえば到達する定義）をすべて集めたものを全体集合とした時，その部分集合のクラスが作る集合束をとる．$\wedge$ と $\vee$ は，それぞれ集合の積と和に対応する．ただし，Kildall が半束を採用したことからわかるように，束の 2 演算のうち本質的には一方だけを用いればよい．式 (1.1) ～ (1.4) に対応してそれを $\wedge$ とすると，問題によって $\wedge$ は集合和に解釈されることも（たとえば到達する定義，活きている変数），集合積に解釈されることも（たとえば使える式，忙しい式）ある [18]．

関数 $f \in F$ は問題によって異なるが，たとえば到達する定義の場合は，

$$f_e(x) = x \cap K_e \cup G_e$$

の形をとる．G_e は辺 e に対応するプログラム・コードで生成される定義の集合，K_e はそこで消滅する定義の集合で $\bar{K}_e$ はその補集合である．解 x_v は，頂点 v に対応するプログラムの場所で観測可能なデータを表す．

$f \in F$ の単調性を仮定した場合，これはデータフロー解析の中のかなり広い（しかしすべてではない）問題群をおおう [30]．

[例]

データフロー解析問題の具体例として，最大公約数をとる簡単なプログラムを対象に，“到達する定義” の解析を定式化しよう．図 1.5 に，対象とする Pascal プログラムと対応するフローグラフを示す．

```
program gcd(input,output);
  var a,b,t: integer;
  begin
    read(a,b);
    while a<>0 do begin
      t := a;
      a := b mod a;
      b := t
    end;
    writeln('gcd =',b)
  end.
```

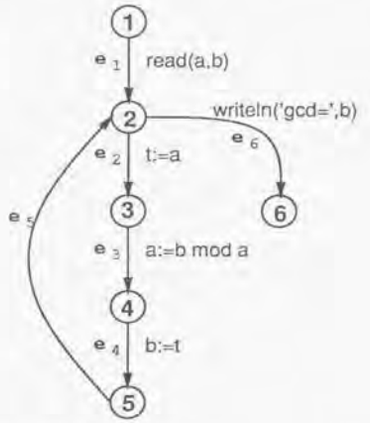


図 1.5 データフロー解析問題例

“到達する定義” における “定義” とは，代入文などの実行によって，プログラム変数に値が結びつけられることをいう．たとえば辺 e_3 の代入文によって，プログラム変数 a に値が定義される．それがフローグラフ上のある頂点に “到達する” とは，その頂点でその定義が参照可能なことをいう．たとえば，e_3 における a の定義は，頂点 4, 5 などに到達する．

変数への値の定義は，プログラムの実行文（辺に対応）とプログラム変数の対で識別できる．そこで，たとえば辺 e_3 における変数 a への値の定義を，$a3$ というように表記することにする．このプログラムのすべての定義の集合 D

を数え挙げると，次のようになる．

$$D = \{a1, b1, t2, a3, b4\}$$

D の部分集合が作るクラス，すなわち 2^D を，集合束とみなしたものが対象とする束 L となる．

ある辺でプログラム変数 x に値の定義が行なわれた時，ここに到達した他の辺における同じ変数 x への値の定義は消滅する．この点を考慮して，式 $f_e(x) = x \cap \bar{K}_e \cup G_e$ を明示的に書き下し，方程式を作ると次のようになる．

$$\begin{aligned} x_1 &= \{\} \\ x_2 &= ((x_1 - \{a3, b4\}) \cup \{a1, b1\}) \cup x_5 \\ x_3 &= x_2 \cup \{t2\} \\ x_4 &= (x_3 - \{a1\}) \cup \{a3\} \\ x_5 &= (x_4 - \{b1\}) \cup \{b4\} \\ x_6 &= x_2 \end{aligned} \tag{1.12}$$

この不動点解と経路解は一致し，次のようになる．

$$\begin{aligned} x_1 &= \{\} \\ x_2 &= \{a1, b1, a3, b4, t2\} \\ x_3 &= \{a1, b1, a3, b4, t2\} \\ x_4 &= \{b1, a3, b4, t2\} \\ x_5 &= \{a3, b4, t2\} \\ x_6 &= \{a1, b1, a3, b4, t2\} \end{aligned}$$

1.2.6 記号実行

記号実行 (symbolic execution) とは，プログラムの入力として具体的な数値を与える代わりに，数値を代表する記号を与えてプログラムを模擬的に実行し，その結果を評価する手法である [74].

この問題は，データフロー解析と同様に，プログラムのフローグラフ上における状態の遷移を解析するものと考えることができる．記号実行の場合，もし

状態がプログラム変数に結びつけられた記号値で表され，この記号値の空間が陽に数えあげられるなら，データフロー解析と同じ構造の問題に帰着する．

しかし，一般にプログラム状態は，プログラム変数と記号値を含む等式のほかに，プログラムの実行経路を定める述語を含む論理式によって表される．その論理式は，プログラムの状態空間をモデルとするような適当な論理系（プログラム言語に現れる述語や関数に関しての公理を含む）に属する．ここで状態空間とは，入力値として与えられる記号とプログラムに現れるすべてのプログラム変数を軸として張られ，それぞれがとりうる値の領域の積空間として構成される（一般に不確定値を含む）．論理式を充足可能とする変数の値の組は，一般に状態の部分空間を構成する．状態空間の部分集合族が作る集合束を考えれば，この場合もわれわれの統一的な問題の定式化に形式的には適合することになる．

束の結びと交わりは，論理式上ではそれぞれ論理和と論理積に対応させるのが自然であるが，データフローの場合と同様に，ここでは交わり $\wedge$ のみが本質的であり，それは論理和に対応する（半束の表現を，データフロー解析における習慣にしたがって $\wedge$ を用いて記述することにしたが，この場合は $\vee$ を用いた方が自然だったことになる）．また，順序 $\leq$ は，含意に対応する．

各辺の上の変換関数は，プログラミング言語の文の意味を定義する意味関数に対応する．

しかし，このような一般的な枠組では，状態間の順序関係の判定に定理証明機能を必要とするので，一般に難しい．そこで，対象とするプログラムの性質を限定し，プログラムの状態に対応する論理系を扱いやすいものにすることが考えられる．たとえば，述語として等号関係のみを扱うとか [36]，半順序関係のみを扱う [48] という方法がある．また，記号実行の過程で得られる，状態を表す論理式を，何らかの基準で弱めることにより，プログラムの繰り返し構造に対する不変表明を発見する試みもある [59, 60, 48].

[例]

データフロー解析で対象としたプログラム図 1.5を，ここでも対象とする．記号実行では，プログラムの入力に対して記号値を与える．このプログラムの場合，入力となるのは最初の read 文による変数 a, b への値の読み込みであるから，このとき a, b に与えられる値をそれぞれ α, β と表すことにする．一般に，

関数や手続きに対して記号実行を行う場合は，引数や大域変数を通じて入力が与えられるので，それらに記号値を割り当てることになる．

記号実行が扱うプログラムの状態は，プログラム変数の値を定める部分と論理的な制約条件を定める部分からなる論理式として表される．すなわち，BNF 風に書けば，

＜プログラム状態＞ ::= ＜値定義＞ ∧ ＜制約＞

プログラム状態 S にたいし，その値定義部分を取り出したものを S_V，制約部分を取り出したものを S_C と表すことにする．なお，この記号実行の例題説明に限り，∧ と ∨ をそれぞれ連言記号および選言記号として用いる．

プログラム変数に割り当てられる値は，記号値式として与えられる．すなわち，

＜値定義＞ ::= ＜値定義式＞(∧＜値定義式＞)*
＜値定義式＞ ::= ＜プログラム変数＞ = ⊥ |
＜プログラム変数＞ = ＜記号値式＞

ここで ⊥ は不確定値である．記号値式は，記号値定数（今の場合 α, β）およびプログラムに現れる定数と関数記号からなる項である．値定義では，プログラムに現れるすべてのプログラム変数に対し，ちょうど 1 つの値定義式が存在する．また制約は一般の論理式であるが，そこに現れるすべての項も同じく記号値式である．

プログラム変数を含む論理式に対し，与えられた値定義を代入することによって，プログラム変数を陽に含まない論理式に変換することができる．論理式 P にプログラム状態 S の値定義を代入した結果を，$(P)_{S_V}$ と書くことにする．

プログラムの実行がフローグラフ上のある頂点に達したとき，そこで一つのプログラム状態が成り立つ．可能な複数の実行経路を考えたときは，それぞれに応じたプログラム状態の論理和をとることになる．これをプログラム一般状態と呼ぼう．

フローグラフの各辺では，プログラム状態が変換される．その変換はプログラム言語の意味 (semantics) に依存するが，Pascal のような言語では次のようになる．

1. 代入文などでプログラム変数に式の値が割り当てられる場合，その式に現れるすべてのプログラム変数を，現在のプログラム状態の値定義による記号値式に置き換え，それと代入されるプログラム変数を等号で結んだ値定義式を値定義に連言的に加え，もとの値定義から同じプログラム変数に対する値定義式を削除する．値定義 V に 1 つの値定義式 E を加え，対応する古い値定義式を削除した結果得られる新しい値定義を $V \oplus E$ と表すことにする．

2. while 文, if 文などで条件が定められた場合，その条件式に現れるすべてのプログラム変数を現在の値定義による記号値式に置き換え，それを制約部に連言的に加える．

プログラム一般状態に対しては，それを構成する個々のプログラム状態に上の操作を個別に適用し，その結果の論理和をとる．

変換操作を具体的に示すと，次のようになる．

辺 e_1 ではデータの読み込みにより記号値が割り当てられるので，値定義に

$$a = \alpha \wedge b = \beta$$

が与えられる．またこのとき，プログラムの仕様として定められた，入力値に対する入力条件が制約に加えられる．この場合，

$$\mathrm{integer}(\alpha) \wedge \mathrm{integer}(\beta) \wedge \alpha \geq 0 \wedge \beta \geq 0$$

となろう．

辺 e_2 では，while 文の繰り返し条件が成立していることから，制約に

$$(a \neq 0)_{x_1, V}$$

が加えられる．ただし，x_1 は頂点 1 のプログラム状態である．また，代入文の実行により値定義に

$$t = (a)_{x_1, V}$$

が加えられ，t についての元の記号値式は除かれる．

以下，他の辺についても同様．その結果，方程式は次のようになる．

$$
\begin{aligned}
x_1 &= (a = \bot \wedge b = \bot \wedge t = \bot) \wedge \mathrm{true} \\
x_2 &= x_1 \wedge (\mathrm{integer}(\alpha) \wedge \mathrm{integer}(\beta) \wedge \alpha \geq 0 \wedge \beta \geq 0) \oplus (a = \alpha \wedge b = \beta) \vee x_5 \\
x_3 &= x_2 \wedge (a \neq 0)_{x_2,V} \oplus (t = (a)_{x_2,V}) \\
x_4 &= x_3 \oplus (a = (b \bmod a)_{x_3,V}) \\
x_5 &= x_4 \oplus (b = (t)_{x_4,V}) \\
x_6 &= x_2 \wedge (a = 0)_{x_2,V}
\end{aligned}
\tag{1.13}
$$

ただし，ここで x_i がプログラム一般状態を表しているとすれば，新たな値定義式を $\oplus$ によって加えたり，新たな制約を $\wedge$ によって加えたりする操作は，一般状態を構成する各プログラム状態に個別に適用される．上式は，その操作を略記したものと解釈する．

各 x_i は論理式を表しているが，束としての解釈は，これらの論理式を充足する (α, β) 空間あるいは (a, b, t) 空間，あるいは両者の積空間がつくる集合束とみなすことになる．

この方程式の解が求められれば，プログラムの出力時に成立する出力条件が x_6 として得られる．とくに，このプログラムの出力 b の記号値は，x_6 の値定義部から得ることができる．しかし，一般にプログラムが繰り返しを含む場合，不動点解を明示的に求めることは難しい．通常，記号実行の技法では，次のいずれかの方法をとることが多い．

1. 実行経路を指定する．1つの経路に沿った解を得ることは機械的にできる．ただし，式の簡約化機能があることが望ましい．

2. 繰り返しに対し，ループ不変表明を仮定する．ループ不変表明とは，繰り返しの中の1点，たとえば頂点2に対し，

$$x_2 \supset I_2$$

を満たす論理式 I_2 で，プログラムのループ，この場合はフローグラフの閉路 e_2, e_3, e_4, e_5 に沿った変換によって不変性を保つもの，すなわち，

$$f_5(f_4(f_3(f_2(I_2)))) = I_2$$

が成り立つものをいう．

I_2 は x_2 より弱い表明であるが，プログラムの仕様として出力時に成立すべき条件 R が I_2 から導かれれば，プログラムがその仕様を満たすことを検証するのに有効である．この場合，

$$R \equiv b = \gcd(\alpha, \beta)$$

であり，

$$f_6(I_2) \supset R$$

が証明できればよい．そのためには

$$I_2 \equiv \gcd(a, b) = \gcd(\alpha, \beta)$$

とするのが適当である．ただし，実際に検証を行うためには関数 gcd に関する性質を公理として与える必要がある．

1.2.7 高速自動微分法

計算手続きの与えられている関数に対し，各変数についての偏導関数を，やはり計算手続きとして生成する手法を，高速自動微分法 [27, 67] という．

n 個の変数 $z_1, z_2, \ldots, z_n$ から関数 $g(z_1, \ldots, z_n)$ を計算する手続きが与えられているとする．その計算過程から，次のような計算グラフを作る．ここで，計算は，四則演算や初等関数などの単項または2項演算の組合せから成るとする．

計算過程に現れる入力変数 $z_1, z_2, \ldots, z_n$，定数，中間（および最終）結果を置く中間変数，それぞれに対応する頂点からなる頂点集合を作る．各中間変数の頂点に対し，それを終点とし，その値を計算するのに必要な他の変数または定数を表す頂点を始点とする，辺をもうける．各辺 e に，その終点の値を得る演算に対して，その始点の変数による偏導関数 d_e をラベルとして付ける．

情報を表す集合 L としては，頂点に対応する変数および定数と辺に対応する関数から，自由生成される項を考え，それらの項の1つ以上の和をとったものとして生成される式からなる集合をとる．L を半束に対応させたときに，$\wedge$ 演算に相当するのが和になるが，和に対しては冪等則が成り立たないので，半

束にならない．したがってまた，∧演算から導入される半順序も定義できない．しかし，各辺に与える関数 f_e を

$$f_e(x) = d_e \cdot x$$

で定義すれば，問題を表す等式として，式 (1.4) がそのまま使える．たとえば，偏導関数 $\partial g/\partial z_i$ を求める場合，出発点 s は z_i に対応する頂点とし，値 b_s としては 1 を与える[1]．解は，中間変数 w に対応する頂点に対しては，その z_i による偏導関数 $\partial w/\partial z_i$ を与え，関数値 g を与える変数に対応する頂点に対しては，$\partial g/\partial z_i$ を与える．

第 3 章で述べる逐次型解法は，束の半順序性を計算の制御に用いているので，そのままでは使えないが，この問題ではグラフが無閉路となるので，あらかじめ定められた順序により計算すれば良く，アルゴリズムはむしろ簡単になる．

[例]

関数

$$g(z_1, z_2) = \frac{z_1 e^{z_2}}{1 - \sqrt{z_1 z_2}}$$

を対象として考える．対応する計算グラフは，図 1.6のようになる．

$\partial g/\partial z_1$ を求めるには，z_1 に対応する頂点を出発点とする問題を解けばよく，その結果，

$$\partial g/\partial z_1 = \frac{1}{v_5} \cdot v_1 + (-\frac{g}{v_5}) \cdot (-1) \cdot \frac{1}{2v_4} \cdot z_2$$

となる．右辺の 2 つの項が z_1 に対応する頂点から g に対応する頂点に至る経路によって生成されることがわかる．同様に，$\partial g/\partial z_2$ は，z_2 に対応する頂点を出発点とする問題を解くことによって，

$$\partial g/\partial z_2 = \frac{1}{v_5} \cdot z_1 \cdot v_1 + (-\frac{g}{v_5}) \cdot (-1) \cdot \frac{1}{2v_4} \cdot z_1$$

となる．なおここで，

$$v_1 = e^{z_2}$$

[1] この場合，厳密にいえば，z_i に対応する頂点から到達可能な部分グラフを対象として考えている．計算グラフ全体を対象と想定する場合は，形式的に方程式 (1.2) の形で考え，$z_j (j \neq i)$ に対応する頂点 v には b_v として 0 を与えればよいが，本質的には同じである．

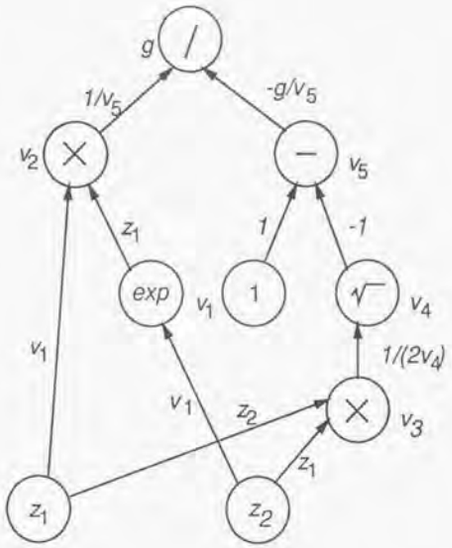


図 1.6 計算グラフ

$$\begin{aligned} v_2 &= z_1 e^{z_2} \\ v_3 &= z_1 z_2 \\ v_4 &= \sqrt{z_1 z_2} \\ v_5 &= 1 - \sqrt{z_1 z_2} \end{aligned}$$

以上の他に，AI における探索問題 [37]（これについては，3.6.5 節で触れる）なども，ここで述べた枠組に含められる．

第 2 章

定式化の比較

これまで述べてきたように，ここで扱っている問題の構造に関する研究には，多くの流れがある．その主なものは，有限オートマトン，最短路問題，データフロー解析であり，またそれらとの関連という意味での連立一次方程式系である．

各々の分野でより一般化した定式化が工夫され，連立一次方程式系との関係も論じられてきた．それらを大別すると，次のようになる．

1. 正規表現によるもの
2. 半環によるもの
3. 束とその上の関数族によるもの

以下でこれらの定式化の方法を述べ，われわれの方法との比較を行なう．

2.1 正規表現による定式化

Kleene は有限オートマトンと正規表現で規定される言語の等価性を示したが，それを自然に発展させ正規表現上の代数によって問題を定式化することができる．これは，Backhouse & Carré[6] や Tarjan[55] でとられている立場であり方法である．とくに，これらの問題がグラフの経路の数え挙げに帰着するという本質を，よく示す定式化といえる．

すでに1.2.4節において，正規言語を受理する有限オートマトンが，われわれの束とその上の関数による定式化によって表現できることを示した．以下の議論は1.2.4節の説明と一部重複するが，改めて正規表現から出発して問題を定式化するという立場から述べる．

アルファベット Σ 上の正規表現を考える．正規表現は，次のように構成的に定義される．

R1 ϵ(空列を示す)，$\emptyset$(空集合を示す)，および Σ の任意の要素は，それぞれ正規表現である．

R2 R_1 と R_2 が正規表現の時，$R_1 \cdot R_2$ および $R_1 \cup R_2$ は正規表現である．R が正規表現の時，R^* は正規表現である．

正規表現は，Σ を文字とする文字列からなる集合の部分集合（この部分集合を言語と呼ぶ）に写像される．その写像を与える規則は，次のようである．

R1' $\epsilon, \emptyset, a \in \Sigma$ は，それぞれ $\{\epsilon\}, \emptyset, \{a\}$ に写像される．

R2' 正規表現 R_1 と R_2 が写像された言語を L_1, L_2 とすると，$R_1 \cdot R_2$ は $L_1 \cdot L_2$ に，$R_1 \cup R_2$ は $L_1 \cup L_2$ に写像される．ただし，$L_1 \cdot L_2 = \{a \cdot b | a \in L_1 \ \& \ b \in L_2\}$．ここで文字列に対する演算 $\cdot$ は，連接である．

正規表現 R が写像された言語を L とすると，R^* は L^* に写像される．ただし，$L^0 = \{\epsilon\}$, $L^i = L^{i-1} \cdot L$ として，$L^* = \bigcup_{k=0}^{\infty} L^k$．

2つの正規表現 R_1 と R_2 は，その写像先の言語が集合として等しいとき，等価であるといい，$R_1 = R_2$ とかく．さらに，正規表現には，次のような等式が成り立つ．

任意の正規表現 R, S, T に対し，

r1 $R \cup S = S \cup R$ （$\cup$ の可換則）

r2 $R \cup (S \cup T) = (R \cup S) \cup T$,
$R \cdot (S \cdot T) = (R \cdot S) \cdot T$ （$\cup$ と $\cdot$ それぞれの結合則）

r3 $R \cdot (S \cup T) = R \cdot S \cup R \cdot T$,
$(S \cup T) \cdot R = S \cdot R \cup T \cdot R$ （$\cup$ と $\cdot$ の間の分配則）

r4 $\emptyset \cup R = R \cup \emptyset = R$ （$\emptyset$ は $\cup$ の単位元）

r5 $\epsilon \cdot R = R \cdot \epsilon = R$ （ϵ は $\cdot$ の単位元）

r6 $\emptyset \cdot R = R \cdot \emptyset = \emptyset$ （$\emptyset$ は $\cdot$ の零元）

r7 $R \cup R = R$ （$\cup$ についての冪等則）

グラフ $G = (V, E)$ 上の経路は，E をアルファベットとする文字列と見なせる．有限オートマトンと正規言語との関係から示唆されるように，経路の集合は正規表現に対応する．たとえば，グラフ上の1つの頂点から別の1つの頂点に至るあらゆる経路の集合，1つの頂点から他のすべての頂点に至る経路の集合，すべての2頂点間の経路の集合，はそれぞれ正規表現で表される．したがって，経路を列挙し，それらの間で成り立つ性質を求めるタイプの問題を，正規表現によって定式化することは自然であり，また可能である．

実際 Tarjan[55] では，正規表現から対象とする問題領域への写像を定める関数を与えることにより，最短路問題，線形方程式系，データフロー解析，などの問題が，統一的に取り扱えることが示されている．たとえば，最短路問題に対しては，cost と shortest という2つの写像を以下のように定義する．

1. $\mathrm{cost}(\epsilon) = 0$, $\mathrm{shortest}(\epsilon) = \epsilon$;
   $\mathrm{cost}(\emptyset) = \infty$, $\mathrm{shortest}(\emptyset) =$ 経路が存在しない;
   $e \in E$ に対して，
   $\mathrm{cost}(e) = c(e)$, $\mathrm{shortest}(e) = e$.
   ただし，$c(e)$ は，辺 e に与えられた費用（距離）

2. $\mathrm{cost}(R_1 \cup R_2) = \min(\mathrm{cost}(R_1), \mathrm{cost}(R_2))$,
   $\mathrm{shortest}(R_1 \cup R_2) = \textbf{if } \mathrm{cost}(R_1) \leq \mathrm{cost}(R_2) \textbf{ then } \mathrm{shortest}(R_1) \textbf{ else } \mathrm{shortest}(R_2)$;
   $\mathrm{cost}(R_1 \cdot R_2) = \mathrm{cost}(R_1) + \mathrm{cost}(R_2)$,
   $\mathrm{shortest}(R_1 \cdot R_2) = \mathrm{shortest}(R_1) \cdot \mathrm{shortest}(R_2)$;
   $\mathrm{cost}(R^*) = \textbf{if } \mathrm{cost}(R_1) < 0 \textbf{ then } -\infty \textbf{ else } 0$,
   $\mathrm{shortest}(R^*) = \textbf{if } \mathrm{cost}(R_1) < 0 \textbf{ then }$ 最短経路が存在しない $\textbf{else } \epsilon$.

2.2 半環を用いた定式化

最短路問題を出発点として問題を抽象化している例では，代数系として半環を用いているものが多い．たとえば，Aho, Hopcroft & Ullman[2], Carré[8], Rote[42], 後藤 [68], などの例がある．

正規表現についても，(r1)~(r7) のような等式を公理として与えて，代数系として定義することもできる．ただし，等式の代入に関する推論規則のみを前提とした公理系では正規表現を完全には公理化できないことが知られており，なんらかの推論規則を付け加える必要がある [44]．

たとえば Backhouse & Carré[6] は，(r1)~(r7) に加えて，1 項演算 $*$ に関する公理を，次の 2 つの等式によって与えた上で，

r8 $R^* = \epsilon \cup R \cdot R^*$

r9 $R^* = (\epsilon \cup R)^*$

次の推論規則を与えている．

[推論規則] 代数系の要素 R が確定的であるとは，任意の要素 x に対し，$x = R \cdot x \rightarrow x = \emptyset$ が成り立つことであると定義する．推論規則は，R が確定的な時，

$$x = R \cdot x \cup S \rightarrow x = R^* \cdot S$$

なお，この公理系は Salomaa[44] が与えたものと，ほぼ同じものである．

同様に，主な発想を，最短路問題と連立一次方程式系との対比から得て，問題の構造を抽象化するような最小の公理群を与えた代数系を定義しようというのが，この立場である．

基本的には，次のような半環が用いられる．代数系 S が，2 つの 2 項演算 $(\oplus, \cdot)$ に関して閉じていて，次式を満たす．

s1 $a \oplus b = b \oplus a$ ($\oplus$ の可換則)

s2 $a \oplus (b \oplus c) = (a \oplus b) \oplus c$,
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$ ($\oplus$ と $\cdot$ それぞれの結合則)

s3 $a \cdot (b \oplus c) = a \cdot b \oplus a \cdot c$,
$(b \oplus c) \cdot a = b \cdot a \oplus c \cdot a$ ($\oplus$ と $\cdot$ の間の分配則)

s4 $a \oplus 0 = a$ (0 は $\oplus$ の単位元) [1]

s4' $0 \cdot a = a \cdot 0 = 0$ (0 は $\cdot$ に関して吸収的)

s5 $1 \cdot a = a \cdot 1 = a$ (1 は $\cdot$ の単位元)

想定している問題により，さらにいくつかの公理が加えられる．正規表現の代数系との異同を見ても分かるように，よく加えられる重要な等式は，次の冪等則である．

s6 $a \oplus a = a$ ($\oplus$ についての冪等則)

ただし，一次方程式系では冪等則は成立しない．

最短路問題では，S を (非負) 整数または実数とし，$\oplus$ を min に，$\cdot$ を $+$ (和) に対応させる．このとき，$\cdot$ に関して可換則が成り立つので，これを公理に加えている例もあるが (たとえば [8])，この可換性は，通常，本質的な役割を果たさない．

問題は，1 項演算 $*$ をどう導入するかである．$a^* = \bigcup_{i=0}^{\infty} a^i$ として導入した場合，右辺の収束性をどう保証するかという問題が生じる．右辺の存在を公理として仮定してしまうという立場も，ありうる．あるいは，すでにあげた Backhouse & Carré のような形の公理を与える方法もある．

半環の公理の一部を除いて，さらに一般化しようとする試みもある．たとえばある種の電気回路の配線問題を扱うのに，Lengauer & Theune[34] は，$\cdot$ の結合性や，$\oplus$ に関して代数系が閉じているという条件を緩めることについて，論じている．

2.3 束を用いた定式化

束を用いている例には，一般化情報ネットワークとして電気回路網上の諸問題，輸送問題，信号フローグラフ，グラフの連結性問題，ゲーム理論，などを統合化した伊理 [24] の例と，データフロー解析の諸問題を半束を用いて定式化した Kildall[32] 以来の方法 ([29, 30, 40, 53]) とがある．

[1] われわれの定式化との対応では，$\oplus$ は $\wedge$ にあたるので，0 と 1 の役割が反対になっている点に注意

本論文でとっているアプローチも，この束を用いた定式化に属する．その特徴は，“情報”を記述する代数系として（半）束をとり，それに対する操作として束から束への写像を与える関数族をとるという形で，対象を2つに分けて扱っているところにある．伊理 [24] では，束の代数構造を規定する $\vee$ と $\wedge$ を内部演算と呼び，本論文で関数空間 F として取り扱っている演算を外部演算と呼んで区別している．

定式化の詳細についてはすでに述べたので，ここでは他の定式化との関係を議論する．

われわれの定式化で関数空間 F を代数系としてみた時，半環をなすことはすでに述べた．したがって，F と2.2節の定式化で対象としている半環とが対応することは，みやすい．この場合，L の構造については，その任意の要素 a を，a を値とする定数関数とみなして，F に埋め込のだものと見ることができる．

あるいは，半環を用いた方法を，われわれの定式化における関数 $f \in F$ を，次の形に制限したものとみなすこともできる．

$$f(x) = a \cdot x$$

ここで，a は半環に属する適当な定数である．

正規表現による方法も，それが対象としている問題の情報構造を表しているという視点から見れば，半環によるものと同じであり，その代数構造も細部の違いしかない．したがって，上に述べた半環による方法と束による方法との対応と同様なことが，正規表現による方法との関係でもいえる．ただし，Tarjan [55] では，正規表現から問題領域への写像という概念を導入し，その写像のとり方により，本論文の定式化が F に与えているような柔軟性をもたせている．

2.4 考察

本論文でとった方法の根拠について，他の方法との比較を通して考察する．

問題の記述を，方程式 (1.1)~(1.4) あるいは (1.5) によるものとすると，そこでもっとも基本的な演算とその性質は，L における演算 $\wedge$ とその可換性および結合性，F における演算 $\cdot$ とその結合性であることが分かる．次の3つの概念は，これらに続いて基本的と思われるが，問題を方程式として記述するためだけなら，とくに必要とはしないわけである．

1. 正規表現で基本的な演算である $*$ 演算．
2. 半環としてみた時に基本的な法則となる $\wedge$ と $\cdot$ の分配性．
3. （半）束としての L における冪等則，あるいは同じ意味で半順序性．

このうち $\wedge$ と $\cdot$ の分配性は，われわれの記法では関数 $f \in F$ の分配性に対応するが[2]，たとえば Kam & Ullman[30] が示すように，データフロー解析の問題の中には単調性は満たすが分配性を満たさないものが存在する．また，ある意味で有限オートマトンを拡張したとみなせる Milner の CCS では [35] では，われわれの $\wedge$ と $\cdot$ に相当する2つの演算（Milner の記法で . と +）の間に必ずしも分配性が成り立たないという観察を理論の出発点にしている．

正規表現の $*$ 演算と束の半順序性は，問題の解を求めるのに必要となる概念である．前者は消去型の解法で，後者は逐次型の解法で用いられる．いずれも，対象とするグラフが閉路をもつ時に，はじめて必要とされる．実際，もし閉路が存在しなければ，あらかじめ定められた順序にしたがって計算することにより，方程式の解は代入操作のみで求められ，$*$ 演算も半順序性も必要としない．たとえば，高速自動微分法はこれに該当する．

線形方程式系の場合は，L に冪等性が成り立たない．x_0 と真の値との間の距離の上限を評価することにより，順序関係が導入できるが，有限下降鎖条件はもちろん成り立たない．これに対しては，周知のように，適当な誤差範囲で反復を打ち切るという措置がとられる．

以上の考察を前提として，本論文でとっている定式化の特徴を挙げると，次のようになる．

双対性 上に述べたように，この問題のもっとも基本的な構成要素は $\wedge$ 演算と $\cdot$ 演算であるが，両者を半環として一つの代数系を構成する2つの演算と見るよりも，本論文におけるようにある種の双対性をもつ2つの空間（1つは情

[2] われわれの定式化で，F における演算 $\wedge$ と $\cdot$ の間には分配則が成り立つが，これと関数 $f \in F$ の分配性とは異なる概念である．しかし，半環を用いて定式化した場合にそこで成り立つ分配則は，後者に対応する．

報を表す空間，もう1つは操作を表す空間）のそれぞれに所属させた方が，問題の本質をより明確に表すように思われる．

その一つの根拠は，すでに述べたように，$\wedge$ と $\cdot$ の分配則が，必ずしも基本的な性質とはいえない点にある．

さらにより重要な論点は，この両者の演算と，対象とする問題のグラフの位相的な構造との関係である．$\wedge$ 演算の定義されたデータ空間の要素は，グラフの頂点に結びつけられており，$\cdot$ 演算の定義された関数空間の要素は，辺に結びつけられている．比喩的にいえば，データは頂点と頂点を結ぶ辺の上を流れ，その間に変換を受けると見なせる．$\wedge$ 演算は，頂点の回りで，そこに入ってくる辺の上のデータの流れの集合に対して適用される演算を表し，$\cdot$ 演算は，辺の列としての経路に沿って，その上の操作を結合する演算を表す．また，前者はグラフの辺の並列結合に，後者は直列結合に対応すると見ることもできる．

このように，問題領域の2種類の対象間の関係と，グラフの頂点と辺（位相幾何学的に言えば，0次元単体と1次元単体）との関係が対応する事例は，電気回路における電位と電流との関係に代表されるように，自然界に多く見られる．したがって，データ空間と関数空間，$\wedge$ 演算と $\cdot$ 演算を，位相的な0次元要素と1次元要素との双対関係と対応させ，全体の構造を双対的に捉えることは，重要だと思われる．

代数系としての性質　半環を基本として，対象とする問題の構造を公理的に規定しようとするアプローチでは，場合によってその公理群の取り方が恣意的になる可能性がある．実際，同じような問題を対象としながら，研究者によって公理の取り方に微妙な違いが認められる．

それに対し，束は正規表現と同様に，その代数構造がすでによく研究され，その具体例も多く知られている．さらに，問題に応じて仮定すべき性質も，たとえば，束に関しての完備性，有限下降鎖性や，束から束への関数に関しての単調性，分配性，連続性，など，データ空間と関数空間とに分離され，それぞれで仮定の強さの階層が明確にできる．

"非線形性"　半環を基本とする定式化は，われわれの記法における F の要素を，

$$f(x) = a \cdot x$$

の形に制限したものに相当することを述べた．線形方程式系との対応という意味では，この制限は自然であるが，次章以降で述べる逐次型および消去型の解法では，この"線形性"の仮定はとくに必要としない．

実際，データフロー解析の問題は，このような線形関数ではうまく記述できない（[42] ではそれが試みられているが，きわめて部分的な定式化でしかない）．また，方程式系との対応でいえば，非線形方程式に関しても，ある種の収束条件の仮定のもとで第3章における逐次型の解法が使えるし，非線形関数の逆関数が計算可能であるという条件のもとで，第4章の消去型の解法が使える．すなわち，本論文における定式化は，単により一般的であるというだけでなく，実用的な意味をもつ．

$*$ 演算の扱い　正規表現における $*$ 演算に相当する操作は，天下り的に与えるのでなく，後の第4章で示すように，構成的に導入するという立場をとった．その理由は，次のようである．

- $*$ 演算は問題の記述に必要がないだけでなく，逐次型の解法でも演算として必要としない．
- $*$ 演算は消去型の解法では基本的な役割を果たすが，その操作を構成的に導入した方が，方程式系の解法との関係が明確になると思われる．

半束と束　Kildall を初めとするデータフロー解析の統合的定式化では，束ではなく半束を用いている．確かに，問題の記述と解法には，束の2演算の内の一方しか用いない．本論文であえて束を採用した理由は，次のようである．

- 対象とする問題の情報の構造自身に，束の性質が本来的に備わっていることが多い．たとえば，最短路問題では，2つの値の最小値をとる min 演算のみを用いるが，その双対演算としての max も，対象となる整数や実数の集合上では自然に定義される．また，最短路問題で min 演算を max 演算に置き換えれば，最長路問題という意味のある問題に変換される．

  データフロー解析でも，情報を表す空間は集合束として捉えられ，束の2演算に集合の和と積が対応する．一つの解析問題としては，集合和か集合積のどちらか一方をとることになるが，問題によってそれが和であったり積であったりする．

- 1つの頂点に入る辺の集合に対して適用される演算として $\wedge$ を取ったとして，辺に結びつけられた関数 f_e は，$f_e(x) = a_e \vee x$ の形を取るような問題も存在する（ここで，a_e は束の要素である適当な定数）．前章で挙げた例では，ネットワークの信頼性問題が，これに相当する．この場合は，すでに述べた $\wedge$ と $\cdot$ の間の2つの空間にまたがる双対性が，1つの空間（束）の自己双対性に帰着される．

  このような問題例は案外少ないが，データフロー解析のように，関数 f_e の定義に $\vee$ と $\wedge$ の両者を用いている例もある（1.2.5 節参照）．

- 本論文では本格的な議論の展開は行なわないが，束の両演算を対等な関係で用いることにより定式化ができるような問題群が存在する．伊理 [24] によって取り上げられている2人ゲームなどが，その例である．その定式化については，第 6.3 節で AND/OR 型の問題として述べる．

計算の粒度 伊理 [24] では，束とそれに対する操作を基本として，一連の問題に対する精緻な理論が構築されている．本論文の立場はこれに近いが，多少の違いがあるとすれば，束で表された情報の操作に基づく計算の処理単位の捉え方であろう．

本章で見たいずれの定式化によっても，操作の基本単位は正規表現，半環，あるいは束で表された情報の要素に対するものであり，グラフ上では1つの頂点あるいは辺に，その情報要素と操作が結びつけられている．しかし，これをグラフ全体に対する一括した操作に拡大することも，自然に可能である．このような拡大された操作は，よく行列表現で表される．グラフの辺を，頂点の集合を行および列に対応させた正方行列によって，隣接行列として表すことはよく行なわれるが，その要素を辺に結びつけられた関数に置き換えれば，頂点集合に対応するデータ全体に対する操作が，表現できる．このように，操作の対象となる単位の大きさを，視点によって変化させることができるが，その単位を並列処理などで使われる用語を借用して，粒度 (granularity) と呼ぶことにしよう．

本論文では一貫して，基本となる細かい粒度，すなわち頂点に対応する情報単位を辺に対応する操作単位で処理するという観点を，保持する．もちろん，

行列のような大きな粒度の観点も，理論的な分析の目的では利用するが，アルゴリズムはすべて，基本操作のレベルで記述する．これは，とくに大規模な問題の疎な性質（頂点対の内，辺が存在するものの割合がきわめて低いこと）を利用するには，粒度の細かいレベルの処理から計算を構成していくことが必須だからである．

したがって，たとえば閉路にまつわる処理に対応する $*$ 演算は，関数空間 F の要素となる各関数に対して定義する．この点が，$*$ に相当する演算を集合的な行列演算に対して定義している [24] のアプローチと，異なるところである．

第 3 章

逐次型の解法

前章で定義した問題の解法には，一次方程式系の解法と同様に，逐次型のものと消去型のものとが考えられる．本章では，逐次型のものを取り上げる．

3.1 逐次型アルゴリズム

本章では，束 L は最大元 1 をもち，かつ有限下降鎖の条件を満たすことを仮定する．第 1 章で述べたように，この時 L は完備であり，したがって最小元 0 ももつ．また関数空間 F の関数は，単調性をもつことを仮定する．ただし，場合によっては，より強い条件である分配性を仮定することがある．

問題の形式は，式 (1.4) とする．すなわち，出発点 s があり，その出発点に入る辺が存在しないものと仮定する．1.1.2 節の議論により，この仮定はとくに一般性をそこなうものではない．ここで，式を再掲する．

$$\begin{aligned} x_s &= b_s, \\ x_v &= \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}) \quad (v \neq s). \end{aligned} \tag{3.1}$$

L の演算 $\wedge$ は一般に逆元をもたない．また F の任意の元 f も $f(x) = y$ に対して $x = f^{-1}(y)$ となる逆関数 f^{-1} をもつとは限らない．したがって，一次方程式に対するガウス・ザイデル法のように，あるいは非線形方程式系に対する解法のように，次のような逐次型の収束算法を考えるのは自然である．

【アルゴリズム I1 逐次型基本形】

入力 グラフ $G=(V,E)$，出発点 $s\in V$，束 L，関数空間 F と各 $e\in E$ に対する $f_e\in F$ は，すでに定義した通り．s に初期値 $b_s(\neq 1)$ が与えられているとし，式 (3.1) の形の方程式を対象とする．さらに便宜上，任意の $x\in L$ ただし $x\neq 1$，$e\in E$ につき $f_e(x)\neq 1$ とする[1]．これは，アルゴリズム上で各頂点をすでに調べたことがあるか否かの判定と，その値を書き換える必要があるか否かの判定とを，同時に行うための方便である．一般に元 1 は人為的に導入されるので，この仮定は一般性を失わずに満たすことができる．

出力 V のうち s から到達可能な頂点の集合を V_1，それ以外の頂点の集合を V_2 としたとき，$v\in V_1$ については方程式 (3.1) を満たす解 x_v を得，$v\in V_2$ については $x_v=1$ となる．

内部データ構造 頂点をしまう集合 S を用いる．

手続き

各 $v\in V$ につき $x_v\leftarrow 1$
$x_s\leftarrow b_s$
$S\leftarrow\{s\}$
$S\neq\emptyset$ の間，以下を繰り返す．
　S から任意の頂点を一つ取り出し v とする．
　（S からは v を取り除く．）
　各 $e\in \mathrm{out}(v)$ につき
　　もし $x_{t(e)}\leq f_e(x_v)$ でなければ
　　$x_{t(e)}\leftarrow x_{t(e)}\wedge f_e(x_v)$
　　$t(e)$ を S に加える．

[1] 関数 f が $\forall x,y\in L.\, x<y\rightarrow f(x)<f(y)$ を満たすという意味での強い単調性をもてば，この性質はただちに導かれる．

3.2 アルゴリズムの正当性の証明

アルゴリズム I1 の正当性を，Floyd–Hoare 流の不変表明を用いる方法で証明する．

手続きの 4 行目にある繰り返し終了判定の時点で，不変表明を考える．この時点での V を，次のように，3 つの互いに素な部分集合 V_a，V_b，V_c に分割する．

$$\begin{aligned} V_a &= \{v|v\in S\},\\ V_b &= \{v|v\in V-S, x_v\neq 1\},\\ V_c &= \{v|v\in V-S, x_v=1\}. \end{aligned}$$

補題として，次の 4 つの不変表明が成り立つことを示す．

補題 3.1 $v\in V_b$ で，辺 $e=(v,w)$ が存在すれば，

$$x_w\leq f_e(x_v) \tag{3.2}$$

証明 繰り返しの回数に基づく帰納法による．最初に繰り返しに入る時点では，$V_b=\emptyset$ だから自明に成り立つ．

k 回目に成立しているとして，$k+1$ 回目にも成り立つことを示す．V_b に新たに加わる頂点は，5 行目で取り出される v である．$\mathrm{out}(v)$ に属する e に対し，$w=t(e)$ とすると，8 行目の条件文から，もともと $x_w\leq f_e(x_v)$ が成立しているか，そうでなければ 9 行目の代入文が実行されて，x_w の代入後の値を x'_w と書くと，

$$x'_w=x_w\wedge f_e(x_v)\leq f_e(x_v),$$

が成り立つ．

同じ代入文によって，w に入る別の辺 d の始点 u で V_b に属するものと，w との間で，(3.2) の関係が影響を受ける可能性がある．この場合は，

$$x'_w=x_w\wedge f_e(x_v)\leq x_w\leq f_d(x_u),$$

だから，やはり (3.2) は依然として成立する．ただし，上の最右の不等式で帰納法の仮定を用いた．（証明終り）

補題 3.2 $v \in V - \{s\}$ に対し，

$$x_v \geq \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}) \tag{3.3}$$

証明 同じく帰納法による．最初に繰り返しに入る時点では，$v \in V - \{s\}$ に対し $x_v = 1$ だから成り立つ．

k 回目の繰り返しで成立しているとして，$k+1$ 回目にも成り立つことを示す．9行目の代入文によって (3.3) の関係が影響を受けないことを示せばよい．右辺に現れる $x_{h(e)}$ が代入によって変化したとすると，代入後の値を $x'_{h(e)}$ としたとき，$x'_{h(e)} \leq x_{h(e)}$ であるから，f_e の単調性より，(3.3) はやはり成り立つ．

左辺の x_v が代入により変化したとする．代入後の値を x'_v とすると，

$$x'_v = x_v \wedge f_d(x_u) \geq (\bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)})) \wedge f_d(x_u)$$

ただし，u は k 回目の繰り返しで S から取り除かれた頂点であり，d は u,v 間の辺である．不等式については，帰納法の仮定を用いている．ところで，$d \in \mathrm{in}(v)$，$u = h(d)$ だから，結局 (3.3) に帰着する．（証明終り）

補題 3.3 $v \in V_b$ で，辺 $e = (v, w)$ が存在すれば，

$$w \notin V_c.$$

証明 $v \in V_b$，すなわち $x_v \neq 1$ だから，9行目の代入文が $t(e) = v$ としてすでに実行されている．また，$v \notin S$ だから，v に対して5行目以下の処理がすでに実行されており，その結果 $e \in \mathrm{out}(v)$ に対し $x_{t(e)}$ がこの処理以前に1であったとしても，9行目の代入文が実行されて $\neq 1$ となる（$\forall x \in L, \forall e \in E.\ x \neq 1 \rightarrow f_e(x) \neq 1$ の仮定に注意）．さらに，一般に x_w の変更の可能性があるのは9行目の代入文のみで，その値は単調に減少するので，一度 $\neq 1$ となればふたたび1となることはない．（証明終り）

補題 3.4 $V_a = \emptyset$ のとき，$v \in V_c$ であれば s から v へは到達不可能である．

証明 $V_a = \emptyset$ のとき，ある $v \in V_c$ があって s から v への経路が存在するとする．$s \in V_b$ だから，その経路上に隣接する頂点 v_k，v_{k+1} があって，$v_k \in V_b$，$v_{k+1} \in V_c$ が成り立つ．これは，補題3.3と矛盾する．（証明終り）

以上の4つの表明が補題として成立することを用いて，アルゴリズムの正当性が示される．

定理 3.1 *アルゴリズム II を入力条件を満たす入力によって実行すると，停止した時にはアルゴリズムの出力条件に述べた意味で，問題 (2.1) の解を得る．*

証明 アルゴリズム終了の時点で，上の4つの表明と繰り返しの終了条件 $S = \emptyset$，すなわち $V_a = \emptyset$ が成立している．補題3.4により，V_b は V_1 に，V_c は V_2 に，それぞれ一致する．

補題3.1より，任意の $v \in V_b$，$e \in \mathrm{out}(v)$ に対し，

$$x_{t(e)} \leq f_e(x_v).$$

ここで補題3.3より $t(e)$ も V_b に属することに注意すると，これを次のように読み替えることができる．任意の $v \in V_b$ ただし $v \neq s$，$e \in \mathrm{in}(v)$ に対し，

$$x_v \leq f_e(x_{h(e)}).$$

さらに，$\wedge$ の性質から次式が成り立つ．

$$x_v \leq \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}).$$

一方，補題3.2より任意の $v \in V - \{s\}$ に対し，

$$x_v \geq \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}).$$

したがって，$v \in V_b - \{s\}$ に対し，

$$x_v = \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}).$$

また，明らかに $x_s = b_s$，かつ $v \in V_c$ に対し $x_v = 1$ ゆえ，アルゴリズムの出力条件が成立する．（証明終り）

3.3 停止性の証明

アルゴリズムの停止性はみやすい．1つの $v \in V$ に対し，x_v の値は手続きの9行目の代入文で真に単調減少するから，有限下降鎖の仮定によりこの代入は有限回しか起こりえない．代入にともない，v は S に加えられるが，V は有限集合と仮定したから，S への頂点の追加は有限回しか起こらない．一方，5行目以降の繰り返しは，1回ごとに必ず1つの頂点を S から取り除くから，いつかは必ず繰り返しの終了条件 $S = \emptyset$ が成立する．

3.4 経路解との関係

Kildall[32] は，関数空間 F に属する関数が分配的であれば，彼のアルゴリズムによって計算される解が，極大な不動点解になるとともに，(1.5) を満たすことを示した．同じことは，アルゴリズム I1 についてもいえる．それを見るために，アルゴリズムに仮想的に経路に関する情報を保持するようなデータと処理を加える．

各頂点 v に対し，経路の集合を与える変数 P_v をつくる．すべての P_v は空集合に初期化しておく．ただし，P_s には $\{\epsilon\}$ を与える．ここで ϵ は空列である．

アルゴリズム I1 の繰り返しを，次のように P_v に関する処理をつけ加えるように変更する．

$S \neq \emptyset$ の間，以下を繰り返す．
　S から任意の頂点を一つ取り出し v とする．
　各 $e \in \mathrm{out}(v)$ につき
　　もし $x_{t(e)} \leq f_e(x_v)$ でなければ
　　$x_{t(e)} \leftarrow x_{t(e)} \wedge f_e(x_v)$
　　$P_{t(e)} \leftarrow P_{t(e)} \cup \{p \cdot e | p \in P_v\}$
　　$t(e)$ を S に加える．

ここで，$p \cdot e$ は，経路 p の後ろに辺 e をつけ加えた経路を表すものとする．

F が分配的であれば，この繰り返しの先頭で，次の不変表明が成り立つ．

補題 3.5 次式が，不変表明として成り立つ．

$$x_v = \bigwedge_{p \in P_v} f_p(b_s) \tag{3.4}$$

証明 帰納法による．最初に繰り返しに入る時点では，自明に成り立つ（$\wedge$ の単位元は1だから，空集合の要素に対して $\wedge$ をとったものは1とみなすことに注意）．

k 回目の繰り返しで成立しているとして，$k+1$ 回目にも成り立つことを示す．この間に S から取り出された頂点 v の後続点で，x および P の値が変更を受けたものを w とする（アルゴリズム中では $t(e)$ と表されている）．x_w, P_w の代入後の値をそれぞれ x'_w, P'_w とすると，次式が成り立つ．

$$x'_w = x_w \wedge f_e(x_v) = \bigwedge_{p \in P_w} f_p(b_s) \wedge f_e\Big(\bigwedge_{p \in P_v} f_p(b_s)\Big)$$

$$P'_w = P_w \cup \{p \cdot e | p \in P_v\}$$

ここで f_e の分配性を考慮すれば，計算により

$$x'_w = \bigwedge_{p \in P'_w} f_p(b_s)$$

の成り立つことがわかる．すなわち，(3.4) が任意の頂点に対して相変わらず成立する．（証明終り）

このことから，アルゴリズムが終了した時点でも，式 (3.4) の成立がいえるが，そのときの各頂点 v に対する P_v は，一般に $\mathrm{path}(s,v)$ の部分集合となっている．$\mathrm{path}(s,v) - P_v \neq \emptyset$ の場合，任意の経路 $p \in \mathrm{path}(s,v) - P_v$ について，

$$x_v \leq f_p(b_s) \tag{3.5}$$

がいえれば，経路解と不動点解との一致がいえることになる．ところが，上の関係は第1章で証明した補題 1.1 にほかならない．

これにより，次の定理が成り立つ．

定理 3.2 アルゴリズム *I1* の解は，F が単調性を満たすとき次の性質をもつ．

$$x_v \leq \bigwedge_{p \in \mathrm{path}(s,v)} f_p(b_s)$$

さらに，F が分配性をもつときには，上式の不等号が等号として成立する．したがってこの場合，得られた解は，不動点解の内で最大のものとなる．

この性質はよく知られているが，その根拠としては通常，I1 アルゴリズムと類似する Kildall のアルゴリズムに対し，Kildall 自身が与えた証明 [32] が参照される．しかし，Kildall の証明は不動点解と経路解の関係を明示しておらず，また F が単調な場合と分配的な場合とを対比していない．しかもその証明は，分りやすいとはいえない．ここで述べた証明は，解が経路の部分集合から作られるという構造を明確にしている点からも，意味があると思われる．

なお，Kam & Ullman [30] は，分配性の成り立たないデータフロー問題に対しては，経路解を求める解法が一般には存在しないことを示した．この場合も，逐次解法で求められる極大不動点解は，実用的な意味がある．

3.5 データ構造の選択によるアルゴリズムの多様性

アルゴリズム I1 は，内部データ構造として設定した集合 S の実現方法によって，動作が異なる．たとえば，

1. S をスタックとする場合．深さ優先探索となる．
2. S を待ち行列とする場合．広さ優先探索となる．
3. 優先順位付き待ち行列とする場合．最短路問題のダイクストラ法は，優先順位を，S に登録されている頂点に対し現時点で計算されている最短距離の昇順につけた場合に相当する．ただし，後で述べるように，ダイクストラ法が使える条件は，かなり限定される．
4. S を先頭と後尾の両者を入口とするリストとする場合．最短路問題に対する Pape[38] による方法が実現できる．

なお，スタック，待ち行列やリストとして実現する場合，すでに登録されている頂点を二重に登録することがないように工夫しないと，むだが生ずる．

3.6 分野別の解法との関係

逐次型のアルゴリズムは，個別の分野ですでにさまざまな提案がされている．以下で，アルゴリズム I1 と，それらの既存の解法との関係を述べる．

3.6.1 到達可能性と極大木

グラフの到達可能性を検査する方法がアルゴリズム I1 のベースとなっているともいえるので，I1 と同じ構造をした手法がこの問題に使えることは明らかであり，また実際よく使われる．

なお，第 4 章でグラフの極大木の性質を使うので，ここで極大木についても述べておく．与えられたグラフ（ここでは有向グラフ）は，ある一つの頂点から任意の他の頂点に到達可能という意味で，連結であるとする．そのグラフの閉路をもたない部分グラフで，どの辺の終点にもなっていない一つの頂点が存在し，それを除く他のすべての頂点に入る辺の数はちょうど 1 であるもののうちで極大なもの，すなわちその部分グラフに属さない辺を 1 つでも付け加えればこの性質が満たされなくなるものを，極大木という．極大木を求めるには，到達可能性を検査するアルゴリズムの中で，新たに到達可能と判定された頂点が見つかるたびに，その頂点に達するのに使われた辺を木に属するものとして登録していけばよい．

3.6.2 線形方程式系

われわれのアルゴリズムはガウス・ザイデル法と類似する．ガウス・ザイデル法を問題 (3.1) に素直に適用すると，次のようになる．

[アルゴリズム I2 逐次型ガウス・ザイデル法]
入力 アルゴリズム I1 と同じ．ただし，s を除く頂点の集合 $V - \{s\}$ に適当な順序 S が定められているとする．
出力 アルゴリズム I1 と同じ．
手続き
$x_s \leftarrow b_s$
$change \leftarrow$ 真

change が真の間，以下を繰り返す．
　change ← 偽
　S の順に取り出した頂点 v について，以下を行なう．
　　$x \leftarrow \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)})$
　　もし $x \neq x_v$ なら
　　　$x_v \leftarrow x$; *change* ← 真

この形のガウス・ザイデル法と比べて，アルゴリズム I1 は次のような特徴をもつ．

1. 計算の順序が，変数 x_v に対して固定的でない．
2. 逐次的な変数の値の更新を，式 (3.1) の右辺をそのまま計算するのではなく，変更のあった変数の値の変化分を伝播させるという形にして，計算量を減らしている．

これらは原理的には線形方程式系にも適用可能であるが，問題の構造が行列表現よりもグラフ表現が向くという程度に疎であること，および各変数の値が単調に減少し，収束計算回数が本質的に有限であるだけでなく，その数が少ないこと，が効果をあげる要因となる．

3.6.3 最短路問題

アルゴリズム I1 は，最短路問題でポテンシャル法として知られている方法 [7] と，本質的に同等である．しかし，ここで定式化した一般化された問題に対してこのアルゴリズムが正しく動作することは自明ではなく，実際，正当性の証明は前節で述べたようにかなり複雑となる．

収束条件

最短路問題に関しては，すでに述べたように，有限下降鎖の条件を緩めてもアルゴリズム I1 は収束する．たとえば，束 L として実数 $\cup\{\infty, -\infty\}$ に通常の順序を導入したものをとる．そして，負の距離をもつ閉路がないものと仮定する．

アルゴリズムの停止性は，次のように証明できる．

まず，アルゴリズムによる解と最短路との関係を改めて考える．3.4 節で述べたように，(3.1) の解は，経路上に拡張された変換関数によって，出発点に与えられた初期値を変換した結果を，あらゆる経路についてとった交わりと密接に関係づけられる．最短路問題の場合，その本来の問題の意味からも，出発点からある頂点へ至るあらゆる経路の中で，もっとも距離の短いものの距離に，その解が対応していなければならない．この関係を直接的にみるために，3.4 節と同様に経路に関する情報をアルゴリズム中で保持し処理するようにする．

3.4 節では，各頂点に対し，出発点からその頂点にいたる経路ですでに調べられたものの集合を，変数 P_v に作成した．最短路問題の場合は，束が全順序性をもつために，すでに調べた経路の集合ではなく，その時点で計算されている最短距離を与える一つの経路を保持すれば十分である．そのために，変数 p_v を用意する．p_s は，ϵ に初期化する．アルゴリズムの繰り返し部分に次のように経路についての情報を与える処理を追加する．

$S \neq \emptyset$ の間，以下を繰り返す．
　S から任意の頂点を一つ取り出し v とする．
　各 $e \in \mathrm{out}(v)$ につき
　　もし $x_{t(e)} \leq f_e(x_v)$ でなければ
　　　$x_{t(e)} \leftarrow x_{t(e)} \wedge f_e(x_v)$
　　　$p_{t(e)} \leftarrow p_v \cdot e$
　　　$t(e)$ を S に加える．

このとき，繰り返しの先頭で次の不変条件が成り立つことは，3.4 節と同様の議論により簡単に示すことができる．

$$x_v = f_{p_v}(0)$$

ただし，最短路問題では F が明らかに分配性をもつこと，および $b_s = 0$ を用いている．

また，経路 $p_v = (e_1, \ldots, e_k)$ に対応する頂点列を $v_0, v_1, \ldots, v_k$ としたとき，その部分経路 $p_{(i)} = (e_1, \ldots, e_i), (i < k)$ は，頂点 v_i に対して，少なくとも過去において p_{v_i} に代入されたことがあることは，ただちにいえる．

さて，負の距離をもつ閉路が存在しないことを仮定すれば，p_v に部分的に閉路が含まれないことを示そう．すなわち，経路 p_v に対応する頂点列を $v_0, v_1, \ldots, v_k$ としたとき，$i \neq j \rightarrow v_i \neq v_j$．実際，$i < j$ で $v_i = v_j$ $(= w$ とおく$)$ と仮定する．出発点 s から頂点 w に至る2つの経路 $p_{(i)} = (e_1, \ldots, e_i)$ と $p_{(j)} = (e_1, \ldots, e_i, \ldots, e_j)$ を比べると，仮定から $f_{p_{(i)}}(0) \leq f_{p_{(j)}}(0)$．ところが，$p_{(j)}$ の部分列として $p_{(i)}$ があるということは，頂点 w に対して経路 $p_{(j)}$ が代入された時点で，すでに $p_{(i)}$ が以前に p_w に代入されていることになる．このことは，x_w と p_w の更新の条件と矛盾する．

アルゴリズムがもし停止しないとすると，S への頂点の登録が有限回で終わらないことになる．頂点集合は有限集合と仮定したから，少なくとも一つの頂点 v について，S への登録が無限に行われることになる．そのたびに，x_v と p_v が更新される．x_v の値は真に単調減少し，経路 p_v はそれに対応するから，p_v に時系列的に代入される経路はすべて異なる．辺の集合も有限と仮定しているから，s から v に至る閉路を含まない経路は有限であるので，これは矛盾である．すなわち，アルゴリズムの停止性が証明された．

これを定理として改めて書くと，次のようになる．

定理 3.3 最短路問題において，束 L として実数集合 $+\{\infty, -\infty\}$ をとったとき，もしグラフ上に負の距離をもつ閉路が存在しなければ，アルゴリズム II は停止し，正しい解を与える．

実は最短路問題に限らず，4.2節に示すように，典型的なデータフロー解析問題などの多くの問題では，有限個の閉路を含む経路のみを考えればよい．その時，経路集合の有限性から，束 L に有限下降鎖条件を仮定しなくても，アルゴリズムの停止性が上と同様に示せる．

なお，ここで停止性の証明のために挿入した経路を求めるアルゴリズム上の処理は，実際上も有用である．実用上は，p_v に経路全体でなく v を終点とする経路上の最後の辺をもち，経路全体を s を根とする木として表現すればよい．（しかし，たとえば Lisp などのリストによって経路を実現する場合，そのような考慮は陽に行わなくても，実質的に木構造がつくられる．）

ダイクストラ法

最短路問題に対して有効なダイクストラ法は，次の条件が満たされるときに使うことができる．

1. 束 L が全順序関係をもつこと．
2. 関数 $f \in F$ が，次の意味で拡大的であること．

$$\forall x \in L.\, f(x) \geq x.$$

条件2は，最短路問題にダイクストラ法を適用する場合に仮定する，負の距離が存在しないという条件に対応する．この条件が成立している場合，ダイクストラ法に相当する方法を用いると，アルゴリズム上で，一度 S に登録され，5行目で S から取り除かれて処理された頂点は，それ以降再び S に登録されることがないことが保証できる．

なお，最短路問題に対しても，実用上は，S として待ち行列を用いた広さ優先探索の方法の方が，ダイクストラ法よりも効率がよいことが多い [66]．これは，広さ優先探索により平均的に出発点からの距離が短い頂点が S から選ばれる反面，優先順位付き待ち行列を保持するための余分な処理がなくてすむからである．実際，少し古いデータではあるが，筆者等が行なった実験で次のような結果が出ている．

実験を行なったのは1975年，使用計算機は当時の IBM370/168 である．

問題として与えたグラフの構造は，頂点の数と1点から出る辺の数の平均とをパラメータとして与えて，乱数により作成している．辺の距離も一様乱数で生成した．ダイクストラ法の実現に当たっては，優先順位付き待ち行列をヒープとして作成した．ポテンシャル法では対応するデータ構造を待ち行列とし，広さ優先探索を実行している．この結果からは，効率的にもポテンシャル法の優位性が認められるが，さらにプログラムは単純であり，負の距離も扱えるという利点がある．

その後，同様な趣旨のさらに精緻な実験が行なわれ報告されている [23]．それによれば，やはりヒープによる方法は，平均的に待ち行列を使った場合より効率が悪い．しかし，同じ程度の規模と構造をもつ問題に対する計算時間の分

表 3.1 ダイクストラ法とポテンシャル法の比較

問題	頂点の数	辺の数	ダイクストラ法 (CPU 時間 (sec))	ポテンシャル法 (CPU 時間 (sec))
1	100	565	0.022	0.016
2	100	1036	0.025	0.018
3	300	1662	0.058	0.031
4	300	3089	0.069	0.040
5	400	2253	0.077	0.039
6	400	4286	0.093	0.055

散は小さく，安定している．それに対し，待ち行列は分散が大きい．ポテンシャル法では，初めて登録される頂点はリストの末尾に，すでに登録されたことのある頂点が再び登録される時はリストの先頭におくという，2つの入口をもつ方法 (two-way sequence method [38]) が，平均的に待ち行列より優れ，分散も小さい．ダイクストラ法では，バケット法 [10] を改良した可変バケット法が，平均時間と分散の両面で優れており，2つの入口をもつリストを使ったポテンシャル法より，さらに良い結果を示している．

3.6.4 データフロー解析

一般的に定式化されたデータフロー解析問題には，Kildall [32] が逐次的アルゴリズムを与えている．その後，多くの改良逐次型アルゴリズムが提案された [21, 29, 30, 31, 53].

面白いことに，Kildall より後に発表された既存のアルゴリズムはすべて，x_v の更新を次の形で行っている．

$$x_v \leftarrow \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)}).$$

それに対し，アルゴリズム I1 では，もとの Kildall のアルゴリズムと同様に，

$$x_{t(e)} \leftarrow x_{t(e)} \wedge f_e(x_v),$$

の形で更新している．前者は辺を後ろ向きにみて計算するのに対し，後者は前向きに計算していることになる．両者は，各辺に対して $f_e(x_v)$ の $\wedge$ をとる計算を一度だけ行えばよい場合には，計算の手間が等しくなる．しかし，一般には繰り返しが起こるので，後者の更新方法の方が，変化分のみを計算しているという点で効率がよい．

たとえば，Hecht & Ullman [21] のアルゴリズムは，頂点の計算順序を工夫することによって，Kildall のアルゴリズムより効率をあげようとするものだが，われわれの記法に直せば次のようになる．

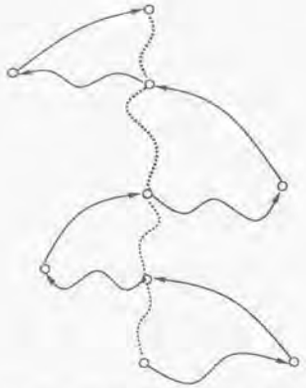
[アルゴリズム I3　逐次型 - 頂点の順序づけ法]
入力，出力はアルゴリズム I1 と同じ．

手続き

頂点を，s を根とする深さ優先極大木の逆後順 (reverse postorder，極大木で定まる半順序で位相整列したものとなる) にまず並べる．この時，s は必ず先頭になる．先頭の s を除いた $V - \{s\}$ の順序を S とする．以下の処理はアルゴリズム I2 とまったく同じ．

Hecht & Ullman は，グラフが簡約可能 (reducible) [2]であればこのアルゴリズムによる手間が f の評価に関して高々 $(d+2)m$ 回，$\wedge$ 演算に関して高々 $(d+2)(m-n)$ 回であることを示した [21]．ここで，n は頂点の総数，m は辺の総数，d は簡約可能グラフの深さである．深さは，グラフ上の閉路を含まない経路に含まれる後退辺の最大数と定義される．後退辺とは，グラフ上にとった極大木に対し，それに含まれない辺で極大木に付け加えると閉路を作るようなものをいう．簡約可能グラフに対しては，この後退辺の定義が，極大木の取り方によらないことが知られている．簡約可能グラフの性質については，第5章で詳しく扱う．なお，深さの直観的な意味は，図 3.1に示す通りである．d はデータフロー解析が通常対象とするような原始プログラムのフローグラフの場合は，1や2などの小さな整数となる．

[2]5.3.1 節参照



深さ 4 の簡約可能グラフの例．ここで点線は極大木に属する経路．横方向の曲線は後退辺を含まない経路．上方向の矢印は後退辺．

図 3.1 簡約可能グラフの深さ

この手間の評価のうち，係数 $(d+2)$ はアルゴリズムの外側の繰り返しの回数に対応する．そのうちの 1 回分は *change* が偽のままであるという収束条件を確かめるためのもので，むだである．アルゴリズム I1 のように，変化のあった頂点を S というデータ構造に登録する機構を作り，収束条件を S が空かどうかで判定するようにすれば，このむだは防げる．d が 1 や 2 の場合はこの効果は大きい．

また，アルゴリズム I3 では外側の繰り返し 1 回ごとに，内側の繰り返しによりすべての頂点について計算処理を行っている．しかし繰り返しの 2 回目以降は，値の変化が伝播して計算処理をする必要が生じるのは，対象とするグラフの閉路の構造に応じた部分グラフに限られる．平均的には，この点を考慮した方が効率がよい．

アルゴリズム I1 では，アルゴリズム I2 との関連でも触れたように，この点に関し 2 つの方法で対処している．

1. 変化のあった頂点を S に登録することにより，計算の及ぶ範囲を限定する．

2. $x_{t(e)} \leftarrow x_{t(e)} \wedge f_e(x_v)$ によって計算することにより，式の中でも変化に応じた分だけ計算するようにしている．

ただし，アルゴリズム I3 の手間の評価を用いるには，アルゴリズム I1 でも深さ優先極大木に基づく頂点順序を利用する必要がある．それには，集合 S を深さ優先極大木の逆後順を保つような優先順位付き待ち行列とすればよい．ダイクストラ法で用いる優先順位付き待ち行列と異なり，この場合の優先順位はあらかじめ定められ固定であるから，その実現はさらに容易である．

この実現方法をとれば，簡約可能グラフに対して，計算の手間は最悪の場合 f の関数評価が $(d+1)m$，$\wedge$ 演算が $(d+1)(m-n)$ となるが，平均的にはこれより十分よくなることが期待できる．なお，他の逐次的な方法と同様に，このアルゴリズムは簡約可能でないグラフにも正しく動作する．

実用的には，深さ優先極大木に基づく優先順位付けを行わず，単純な待ち行列（ただし二重登録は防ぐという措置はつけたもの）を用いても，繰り返しの増大は平均的にはそれほど大きくなく，深さ優先極大木を作る手間や，それに基づく優先順位を保持する手間がないだけ，得なことが多い．

Kildall によるもとのアルゴリズムとアルゴリズム I1 とを比較すると，われわれが S と名づけたリストに，次に計算すべき頂点を登録する代わりに，その頂点の先のすべての辺とその辺のラベルによる計算結果を登録している点が異なる．これは本質的な違いではないが，プログラムの実現上は効率に大きく影響する．

より本質的な違いは，われわれが関数を辺に結びつけているのに対し，Kildall は頂点に結びつけている点である．データを頂点に，関数を辺に結びつけた方が，グラフの位相構造との対応からいっても，最短路問題や他の経路列挙型の問題との対応からいっても，問題の本質が明解になる．実際，Kildall 流の定式化では，頂点の入口と出口の両方でデータを考えるという，わずらわしさが起こる．Tarjan の一連の論文 [53, 55, 56] ではわれわれと同じ形式をとっているので，とくに目新しいことではないが，データフローを扱ったほとんどの論文がこの点で Kildall を踏襲しているので，注意する意味があろう．

3.6.5 探索問題

探索問題を，状態集合と状態遷移のための操作の集合が与えられているものとし，状態集合に含まれる初期状態から出発して，操作を繰り返しながら，あらかじめ定められた状態集合の部分集合である目標状態のいずれかに達する経路を探すものであると定式化すれば，われわれの問題に帰着することや，ここで述べた逐次型の解法が使えることは，ほぼ自明である．しかし，この探索型の問題はきわめてよく現れるので，ここでアルゴリズム I1 を探索問題に適応した形式に表現しておく．

[アルゴリズム I4 逐次型探索]

入力 グラフ $G = (V, E)$．ただし，V は状態集合に対応し，E は状態から状態への遷移を与える操作に対応する．初期状態 $s \in V$，目標状態の集合 $F \subset V$．

出力 s から到達可能な目標状態の 1 つ．

内部データ構造 頂点をしまう集合 S と T を用いる．S の役割はこれまでと同様に，探索過程で到達したがまだその後続点を調べていない頂点をしまうものである．T は，一度到達した状態を登録して，重複して探索しないようにするために用いる．

手続き

$S \leftarrow \{s\}$
$T \leftarrow \{s\}$
$S \neq \emptyset$ の間，以下を繰り返す．
　S から任意の頂点を一つ取り出し v とする．
　各 $w \in \mathrm{succ}(v)$ につき，以下を行なう．
　　もし $w \in F$ ならば，w を解として終了する．
　　もし $w \notin T$ ならば，
　　　w を S に加える．
　　　w を T に加える．

通常の探索問題の場合，状態集合と操作集合があらかじめすべて数えあげられていないことの方が多い．すなわち，グラフ G が初めから陽に与えられているのではなく，任意の状態に対し，適用可能な操作を見つける方法と，その操作で遷移する先の状態を得る方法とが与えられている，というものである．そのような場合でも，逐次型の方法はもちろん有効である．これは，逐次型の方法を，グラフを動的に生成するように解釈することができるからである（後述の消去型の解法では，これは難しい）．

なお，単に到達可能な目標状態を得るだけではなく，そこに至る経路も必要な場合は，最短路問題の節で述べたように，各頂点でそこに至る経路を保存するようにすればよい．

このアルゴリズムでは，一度到達した頂点を T というデータ構造に保持している．アルゴリズム I1 では，この情報を特別に取り扱う代わりに，T に登録される前の頂点は，対応する変数 x_v の値が最大元 1 であることを利用していた．しかし，この探索手法のように，頂点が既探索か否かを陽に取り扱う方が自然であるともいえる．ここでわざわざアルゴリズム I4 を示したのは，この点を比較する意味もある．なお，データ構造の具体的な実現に際しては，T と S を容易に共通化できる．

実際の探索問題には，探索の過程で守らねばならない拘束条件が与えられていることがよくある．また，探索の効率をあげるための工夫が，拘束条件と関連して取り扱われることも多い．AI の教科書によく登場する最良優先探索や A* アルゴリズムはその典型的な例であるが，手法としてはすでに議論した最短路問題の解法（とくにダイクストラ法が援用されることが多い）と同型である．

分枝限定法

数理計画法の分野で，整数計画法の解法として知られる分枝限定法も，その本質は探索である．ここでは，[69] にある分枝限定法の一般的な枠組にしたがって，その方法の骨子がここで取り上げたアルゴリズムと同じ構造であることを示す．

$x = (x_1, x_2, \ldots, x_n)$ を整数変数とし[3]，制約 $x \in D$ のもとで，$z = f(x)$ を

[3] 分枝限定法は，通常，連続変数も含む混合整数計画問題を対象として記述されるが，全整

最小化する問題を考える．この問題に対して，分枝限定法による解法の基本的な構造は，次のようになる．

[アルゴリズム I5 分枝限定法]

入力　目的関数 $f(x)$ と制約 D．このもとの問題を P_* と呼ぶ．

出力　最適解 z

内部データ構造　問題をしまう集合 S を用いる．

手続き

$z \leftarrow \infty$

$S \leftarrow \{P_*\}$

$S \neq \emptyset$ の間，以下を繰り返す．

S から任意の問題を一つ取り出し，P_v とする．

P_v の緩和問題 P_v' を作る．

P_v' の解 $\bar{x}_v$ と対応する目的関数値 $\bar{z}_v$ を求める．

$\bar{x}_v$ を加工し，実行可能解 $\tilde{x}_v$, $\tilde{z}_v$ を作る．

もし $\bar{z}_v < z$ ならば，

$z \leftarrow \tilde{z}_v$,　$x \leftarrow \tilde{x}_v$

P_v の子問題 $P_{v,1}, P_{v,2}, \cdots, P_{v,k}$ を作り，S に入れる．

ここで，緩和問題とは，制約条件の一部を緩和したもの．典型的には変数の整数制約を緩和して実数領域の問題とする．実行可能解とは，制約条件を満たす解．たとえば，実数解を丸めて整数解とするなどが，整数条件を緩和した問題に対する解から実行可能解を得る典型的な方法である．子問題は，たとえば整数条件を満たしていない変数に対して，有界制約を2つに分解して2つの子問題とすることなどが，典型的な作成方法である．

数問題としても本質的には同じなので，ここでは整数変数のみとした．

第 4 章

消去型の解法

本章では，消去型の解法を扱う．ガウス消去法などの消去型のアルゴリズムは，線形方程式の解法としてよく知られる．最短路問題やデータフロー解析問題でも，消去型に分類しうる解法が，それぞれ独立に工夫されてきた．それらと，線形方程式系の消去型アルゴリズムとの関係もすでに指摘されているが，ここでは行列記法とグラフ記法との関係を明確に対応づけながら，系統的にアルゴリズムを導出することを試みる．

4.1　行列記法

線形方程式系の消去法アルゴリズムとの対応をとるために，行列記法を導入する．

$n = |V|$ として，$n \times n$ の正方行列 A を考える．頂点集合 V の要素を，適当な順序で並べる．その頂点順序に従い，頂点を行列 A の行および列に割り当てる．以下で，行（列）i に割り当てられた頂点を，混乱のない限り頂点 i と呼ぶ．

行列の (i,j) 要素に頂点 j から i へ向かうの辺 e に対応する関数 f_e を割り当てる（グラフの隣接行列では，要素 (i,j) に辺 (i,j) を対応させることが多いが，ここではあえて逆向きにしていることに注意）．j, i 間に辺が複数存在する場合は，A の (i,j) 要素 f_{ij} を次式で定義する．

$$f_{ij} = \bigwedge_{\{e \in E \mid h(e)=j, t(e)=i\}} f_e$$

j、i 間に辺が存在しない場合は，F における演算 $\wedge$ の単位元である関数 τ を割り当てる．

変数 x_v を同じ頂点順序に従って並べたものをベクトルと考え，x と書く．同様に，定数 b_v を並べたベクトルを，b と書く．行列 A とベクトル x の積 Ax を次式で定義する．

$$y = Ax \leftrightarrow y_i = \bigwedge_j f_{ij}(x_j).$$

この記法を用いると，基本方程式 (1.1) および (1.2) は，それぞれ次のように書ける．

$$x = Ax, \tag{4.1}$$

$$x = Ax \wedge b. \tag{4.2}$$

ただし，ベクトル間の演算 $\wedge$ を，要素ごとに $\wedge$ をとったものとして用いている．

(4.2) 式は，領域を実数体上に移し，関数適用を係数の乗算，$\wedge$ を加算に読み替えれば，線形方程式 $x = Ax + b$ または $(I - A)x = b$ に対応する．

4.2 消去型アルゴリズムの導出

本章では，束 L は完備束であるとし，また関数空間 F の関数は連続性を持つことを仮定する．

4.2.1 消去法の基本操作

線形方程式 $(I - A)x = b$ の解を求めるには，消去法により $(I - A)^{-1}$ を直接計算するか，$(I - A)$ の三角分解を行って $Ux = L^{-1}b$ (U は上三角行列，L は下三角行列) の形式に変換し，後退代入を行うことが通常である．(4.2) に対しては，減算に相当する $\wedge$ の逆演算と，逆行列を求める際に必要となる除算に相当する関数適用の逆演算とが，一般に存在しない．しかし，消去法の構成を考えれば，以下のような 2 種類の操作により消去法の掃き出し計算が可能なことがわかる．

1. 代入操作： 行 i から変数 x_k を消去．すなわち，行 i の右辺に現れる x_k に，行 k の右辺を代入する．ここで，行 k の右辺には x_k の項が現れないものとする (いいかえれば，$f_{kk} = \tau$)．代入した結果は，

$$\begin{aligned} x_i &= \bigwedge_{j \neq k} f_{ij}(x_j) \wedge b_i \wedge f_{ik}(\bigwedge_j f_{kj}(x_j) \wedge b_k) \\ &= \bigwedge_{j \neq k} (f_{ij} \wedge f_{ik} \cdot f_{kj})(x_j) \wedge (b_i \wedge f_{ik}(b_k)). \end{aligned} \tag{4.3}$$

ここで，関数の連続性の仮定を用いている．

2. 還元操作： 行 k の右辺に x_k が存在するとき，式を x_k について陽に解く．このためには，$f \in F$ に対して，$x = f(x) \wedge a$ を満たす x を $f^*(a)$ として与える関数 f^* が求められればよい．

$$f_k = \iota \wedge f \wedge f^2 \wedge \cdots \wedge f^k,$$

とおき，f^* を次式で定義する．

$$f^*(x) = \inf\{f_i(x) | i \geq 0\}.$$

$y^* = f^*(a)$ とすると，

$$\begin{aligned} f(y^*) \wedge a &= f(\inf\{f_k(a) | k \geq 0\}) \wedge a \\ &= \inf\{f(f_k(a)) \wedge a | k \geq 0\} \\ &= \inf\{f_{k+1}(a) | k \geq 0\} \end{aligned}$$

ここで f_k の定義から，$f_{k+1}(a) \leq f_k(a)$ が成り立つから，上式の最後の項は $\inf\{f_k(a) | k \geq 0\}$ すなわち y^* と等しい．したがって，

$$y^* = f(y^*) \wedge a.$$

なお，$\iota^* = \iota$, $\tau^* = \iota$ である．

f から f^* をつくる操作を用いて，

$$x_k = \bigwedge_j f_{kj}(x_j) \wedge b_k$$

は，右辺に x_k の項が陽に含まれる場合 ($f_{kk} \neq \tau$)，次のように変形される ($\tau^* = \iota$ だから，次式は実は $f_{kk} = \tau$ の場合も含め，一般に成立する)．

$$x_k = \bigwedge_{j \neq k} f_{kk}^* \cdot f_{kj}(x_j) \wedge f_{kk}^*(b_k) \tag{4.4}$$

関数 f に対して f^* を作る演算を $*$ 演算と呼び，f^* を不動点関数と呼ぶことにする．

ここで導入した $*$ 演算は，もともとの方程式 (4.2) の右辺の各行 i に現れる f_{ii} か，代入操作を繰り返した結果生じる f_{ii} に対して適用される．前者はセルフループ（始点と終点が一致するような辺）のラベルであり，後者は代入操作の意味に注意すれば，問題のグラフ上の閉路 p に対して定義される関数 f_p（これは，閉路上の辺のラベル f_e の合成として定義された）に相当することが分かる．これらの関数 f に対して $*$ をとった結果は，多くの場合，有限の k に対する f_k と一致する．すなわち，$f_k = f_{k+1}$ となる k が存在すれば，$f^* = f_k$ である．このような k の最小値を問題の次数と呼べば，一般に次数が高いほど複雑な問題であるといえる．実際，逐次型のアルゴリズムが有限回で収束するのは，次数が有限であることによる．

この分類でいけば，最短路問題は 0 次である．ただし，3.6.3 節で議論したように負の距離を持つ閉路はないものとする．実際，閉路上の距離がかならず非負であれば，$*$ 演算の対象となる関数は，$f(x) = x + d, (d \geq 0)$ となるが，この場合，$x = f(x) \wedge a$ は，$x = \min(x + d, a)$ となり，その解は明らかに $x = a$ となる．すなわち，$f^*(a) = a$，あるいは $f^* = \iota$ である．なお，$f(x) = x + d, (d < 0)$ に対しては，$f^*(a) = -\infty$ と形式的に定めてもよい．

同様に，ネットワークの信頼性問題の次数も，0 である．

データフロー解析の多くの問題（到達する定義，使える式，忙しい式，活きている変数，など）は 1 次である．すなわち，$f^* = \iota \wedge f$ である．これも，関数が

$$f(x) = x \cap K \cup G$$

という形をしていることを考慮すれば，明らかである．ただし，K, G, x はいずれもある集合 D の部分集合のクラス 2^D の要素であり，K と G は定数，x は変数である．なお，データフロー解析問題の中には，2 以上の次数を持つものや，有限の次数を持たないものも知られている [55]．

有限次数の問題に関しては，連続関数 f に対して f^* も明らかに連続である．以下では，一般に f^* の連続性を仮定する．

4.2.2 基本操作のグラフ上での解釈

消去法の基本操作を，グラフ上の変形操作として解釈する．

1. 代入操作

   (4.3) 式は，グラフの接続関係を陽に表すように書けば，次のようになる．

$$x_i = \bigwedge_{j \in \mathrm{pred}(i) \cup \mathrm{upred}(k) - \{k\}} (f_{ij} \wedge f_{ik} \cdot f_{kj})(x_j) \wedge (b_i \wedge f_{ik}(b_k)) \tag{4.5}$$

   (4.5) 式の右辺第 1 項で j を 1 つに固定させて考えたとき，3 頂点 i,j,k の関係は図 4.1のようになる．ここで代入操作は，次のようなグラフの変形に対応する．

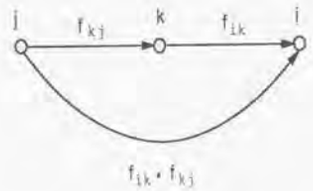


図 4.1 代入操作

   (a) j, i 間に辺がない場合には，j から i への辺をつくり，そのラベルを $f_{ik} \cdot f_{kj}$ とする．すでに，j から i への辺が存在する場合は，そのラベルを $f_{ij} \wedge f_{ik} \cdot f_{kj}$ に変更する．

   (b) k, i 間の辺を削除する．

   このうちの操作 (a) を，以降では “代入 (k, j, i)” と呼ぶ．

2. 還元操作

還元操作は，グラフ上ではセルフループの除去操作にあたる．すなわち，図 4.2のように，k から k へのセルフループがある時，k のすべての先行点 j について以下を行う．

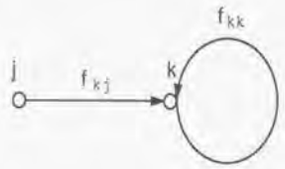


図 4.2 還元操作

(a) j, k 間の辺のラベルを $f_{kk}^* \cdot f_{kj}$ に変更する．

(b) k, k 間のセルフループを削除する．

このうちの (a) の操作を，以降では “前還元 (k, j)” と呼ぶ．

これとある意味で双対な操作として，セルフループのある頂点 k と，その後続点 i に対して，k, i 間の辺のラベルを $f_{ik} \cdot f_{kk}^*$ に変更するというものが考えられる．実際，後で述べるガウス・ジョルダン型のアルゴリズムでは，この操作を用いる．この操作を，“後還元 (k, i)” と呼ぶ．

なお，行列記法を述べた際，2 頂点間に複数の辺がある場合に，それを 1 つに辺に集約する操作を導入したが，これは (1) の代入操作が直列に並ぶ辺をまとめるものであることに対応し，並列に並ぶ辺をまとめるものといえる．この操作は上記の代入操作の中でも，j, i 間にすでに辺が存在する場合には暗に実行されているが，基本操作として独立させた方が見通しがよい．これを合併操作と呼ぶことにする．

3. 合併操作

図 4.3のように，j、i 間に 2 つの辺 e_1，e_2 が存在する場合に，以下を行う．

(a) j, i 間に辺をつくり，そのラベルを $f_{e_1} \wedge f_{e_2}$ とする．

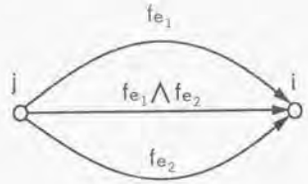


図 4.3 合併操作

(b) e_1, e_2 を削除する．

ただし，以降ではグラフ表現の際でも必要な合併操作は事前に実行されているとし，頂点 u, v 間に辺が高々一つしかないものとする．したがって，辺 e のラベル f_e を，対応する頂点の対 (u, v) により f_{vu} と表記することもある．

4.2.3 ガウス・ジョルダン消去法

まず，ガウス・ジョルダン型の消去法を行列記法で記述する．

[アルゴリズム M1　ガウス・ジョルダン (行列型)]

入力　方程式 $x = Ax \wedge b$ を定める行列 A とベクトル b．

出力　方程式を満たす解 x．

手続き

$x \leftarrow b$ とする．

k を 1 から n まで動かして，以下を行なう．

　j を $k+1$ から n まで動かして，以下を行なう．

　　$f_{kj} \leftarrow f_{kk}^* \cdot f_{kj}$

　$x_k \leftarrow f_{kk}^*(x_k)$

　i を 1 から $k-1$ および $k+1$ から n まで動かして，以下を行なう．

　　j を $k+1$ から n まで動かして，以下を行なう．

　　　$f_{ij} \leftarrow f_{ij} \wedge f_{ik} \cdot f_{kj}$

$x_i \leftarrow x_i \wedge f_{ik}(x_k)$

手続き中の変数 k に対応する行列要素 (k,k) を，かなめ (pivot) と呼ぶ．かなめに応じて，(i,j) で指される範囲に対して実行される操作を，掃き出し演算と呼ぶ．

問題の条件を緩めて，関数 $f \in F$ に関して必ずしも f^* が存在するとは限らないとすると，手続きの中で，f_{kk}^* を求める計算が，実行できないことがある．たとえば，最短路問題で負の距離を持つ閉路が存在する場合である．これを利用すれば，負の距離を持つ閉路が存在しうる場合に，それを検出しながら最短路問題を解くことができる．これは，以下にあげる消去型のアルゴリズムについて共通にいえることであり，逐次型にない特徴である．

このアルゴリズムを，グラフに基づいた表現に書き換えると次のようになる．

[アルゴリズム G1　ガウス・ジョルダン (グラフ型)]

入力　方程式 $x = Ax \wedge b$ に対応するグラフ表現．ただし，(1.2) 式を (1.4) 式に帰着させる操作に対応したグラフ変形を行なう．すなわち，グラフ $G = (V,E)$ に，新たな頂点 s と，s を始点とし $b_v \neq 1$ なる v を終点とする辺の集合 E_s を加えたグラフ $G' = (V \cup \{s\}, E \cup E_s)$．$(s,v)$ 間の新たな辺 e_{sv} には，$f_{e_{sv}}(x) = b_v$ で定義される関数 $f_{e_{sv}}$ をラベルとしてつける．

出力　グラフ $G^* = (V \cup \{s\}, E_s^*)$．ただし，$E_s^*$ は s からすべての $v \in V$ への辺 e_{sv} の集合．方程式を満たす解 x_v は，e_{sv} のラベル $f_{e_{sv}}$ により，$f_{e_{sv}}(0)$ で与えられる．

手続き

/* 以下で還元操作と代入操作を下位手続きとして利用する．手続きの過程でグラフが変形されるが，それにともない pred や succ も更新されることを仮定する．*/

各 $v \in V$ につき，以下を行なう．

もし v にセルフループ e_{vv} があれば，

各 $u \in \mathrm{pred}(v) - \{v\}$ につき，前還元 (v,u) を行う．

e_{vv} を削除する．

各 $w \in \mathrm{succ}(v)$ につき，以下を行なう．

各 $u \in \mathrm{pred}(v)$ につき，代入 (v,u,w) を行う．

v, w 間の辺を削除する．

アルゴリズム M1 では，かなめ k に対して，列は k の右側 ($j > k$ の範囲)，行は全体から第 k 行を除いた範囲を掃き出し演算の対象にしている．ここで，第 k 列にいわゆる積形式の逆行列の列をつくり，さらに掃き出しの対象範囲を列の方も全体から第 k 列を除いた範囲とすれば，よく知られるように逆行列が計算できる．

[アルゴリズム M2　ガウス・ジョルダン逆行列]

入力　方程式 $x = Ax \wedge b$ を定める行列 A．

出力　方程式の解を A^*b によって与える行列 A^*．

手続き

k を 1 から n まで動かして，以下を行なう．

j を 1 から $k-1$ および $k+1$ から n まで動かして，以下を行なう．

$f_{kj} \leftarrow f_{kk}^* \cdot f_{kj}$

i を 1 から $k-1$ および $k+1$ から n まで動かして，以下を行なう．

j を 1 から $k-1$ および $k+1$ から n まで動かして，以下を行なう．

$f_{ij} \leftarrow f_{ij} \wedge f_{ik} \cdot f_{kj}$

$f_{ik} \leftarrow f_{ik} \cdot f_{kk}^*$

$f_{kk} \leftarrow f_{kk}^*$

このアルゴリズムを最短路問題に適用すると，よく知られるように Floyd の方法 [14] になる．この関係をみるには，次の 2 点に注意すればよい．

1. (負の距離を持つ閉路がないことを仮定した) 最短路問題では手続き中

の f^*_{kk} はすべて ι だから，f^*_{kk} が入っている代入文はすべて不要となる．

2. 出力として得られる A^* の第 j 列は，b の第 j 要素を 0，他の要素をすべて 1 とした（単位ベクトルに当たる）場合の解になる．これは頂点 j から他のすべての頂点への最短距離を意味するから，A^* の各要素により全頂点間の最短距離が表されていることになる．

アルゴリズム M2 に対応するグラフ表現は，次のようになる．

[アルゴリズム G2　ガウス・ジョルダン逆行列 (グラフ型)]

入力　方程式 $x = Ax \wedge b$ に対応するグラフ表現 $G = (V, E)$．ただし，アルゴリズム G1 と異なり，b についてはグラフ上で陽に表現しない．

出力　グラフ $G^* = (V, E^*)$．ここで，$E^* \supset E$ であり，E^* の各要素 e に付けられたラベル f_e によって，方程式の解 x_v が，次のように与えられる．

$$x_v = \bigwedge_{e \in \mathrm{in}(v)} f_e(b_{h(e)}).$$

手続き

各 $v \in V$ につき，以下を行なう．
　もし v にセルフループ e_{vv} があれば，
　　各 $u \in \mathrm{pred}(v) - \{v\}$ につき，前還元 (v, u) を行う．
　各 $w \in \mathrm{succ}(v) - \{v\}$ につき，以下を行なう．
　　各 $u \in \mathrm{pred}(v) - \{v\}$ につき，代入 (v, u, w) を行う
　もし v にセルフループ e_{vv} があれば，
　　各 $w \in \mathrm{succ}(v) - \{v\}$ につき，後還元 (v, w) を行う．
　　e_{vv} のラベルを $f^*_{e_{vv}}$ に変更する．

なお，このアルゴリズムの結果得られるグラフ G^* は，元のグラフ G で u から v が到達可能なら (u, v) 間に辺が存在するという，G の閉包になっている．とくに G が強連結の場合は，G^* は完全グラフとなる．

4.2.4　ガウス消去法

ガウス消去法は，線形方程式の解法としてはガウス・ジョルダン法より効率がよい．消去法の掃き出し操作の範囲という点からみれば，ガウス法では，かなめ k に対し行も列も k より大きい範囲を掃き出す．

[アルゴリズム M3　ガウス消去法 (行列型)]

入力　方程式 $x = Ax \wedge b$ を定める行列 A とベクトル b．

出力　方程式を満たす解 x．

手続き

　$x \leftarrow b$ とする．
　k を 1 から n まで動かして，以下を行なう．
　　j を $k+1$ から n まで動かして，以下を行なう．
　　　$f_{kj} \leftarrow f^*_{kk} \cdot f_{kj}$
　　$x_k \leftarrow f^*_{kk}(x_k)$
　　i を $k+1$ から n まで動かして，以下を行なう．
　　　j を $k+1$ から n まで動かして，以下を行なう．
　　　　$f_{ij} \leftarrow f_{ij} \wedge f_{ik} \cdot f_{kj}$
　　　$x_i \leftarrow x_i \wedge f_{ik}(x_k)$

　/* 後退代入 */
　k を $n-1$ から 1 まで動かして，以下を行なう．
　　$x_k \leftarrow \bigwedge_{j>k} f_{kj}(x_j)$

これに対応するグラフ表現は，次のようになる．

[アルゴリズム G3　ガウス消去法 (グラフ型)]

入力　アルゴリズム G1 と同じグラフ $G = (V \cup \{s\}, E \cup E_s)$．

出力　無閉路グラフ $G_U = (V \cup \{s\}, E_U)$．$E_U$ のラベルは後述の “後退代入” により，元の方程式の解を与えるものになっている．

内部データ構造　未処理の頂点をしまっておく集合 W.

手続き

$W \leftarrow V$

各 $v \in V$ につき，以下を行なう．

$W \leftarrow W - \{v\}$

もし v にセルフループ e_{vv} があれば，

各 $u \in \mathrm{pred}(v) - \{v\}$ につき，前還元 (v, u) を行う．

e_{vv} を削除する．

各 $w \in \mathrm{succ}(v) \cap W$ につき，以下を行なう．

各 $u \in \mathrm{pred}(v)$ につき，代入 (v, u, w) を行う．

v, w 間の辺を削除する．

ここで，V から頂点を取り出す順序（または W から取り除く順序）は任意だが，結果として 1 つ定まる．その順序を S とすると，S の最後に s を加えたものは，終了時の無閉路グラフ G_U で定まる半順序（頂点 (v, w) 間に辺がある時 $v \prec w$ とする）で，頂点を昇順に位相整列したものの逆順となる．S の逆順を R とする．

グラフ G_U と順序 R を用いて，次の後退代入手続きにより解が求まる．

[後退代入]

手続き

$x_s \leftarrow 0$

R の順に取り出した v につき，

$x_v \leftarrow \bigwedge_{e \in \mathrm{in}(v)} f_e(x_{h(e)})$

ガウス・ジョルダン法で逆行列を作った場合と同様に，行列を LU 分解した場合の下三角行列 L の逆行列を同時に求めることが，上のアルゴリズムの変形で可能となる．

[アルゴリズム G4　ガウス消去法 (L^{-1} グラフ型)]

入力　アルゴリズム G2 と同様のグラフ $G = (V, E)$.

出力　グラフ $G' = (V, E_U \cup E_L)$．ここで，$E_U \cap E_L = \emptyset$．手続きの中で，アルゴリズム G3 と同様に V の要素の処理順序を任意に定めるが，その順序 S の最後の頂点を t とすると，部分グラフ $G_U = (V, E_U)$ は，t に入る辺が存在しないような無閉路グラフ，部分グラフ $G_L = (V, E_L)$ は，辺の向きをすべて逆向きにしたときに，セルフループを除いて，やはり t に入る辺が存在しないような無閉路グラフとなる．任意の定数ベクトル b に対して，方程式の解は後述の前進代入を部分グラフ G_L に対して行い，その後，後退代入を部分グラフ G_U に対して行ったものとなる．

手続き

$W \leftarrow V$

各 $v \in V$ につき，以下を行なう．

$W \leftarrow W - \{v\}$

もし v にセルフループ e_{vv} があれば，

各 $u \in \mathrm{pred}(v) \cap W$ につき，前還元 (v, u) を行う．

e_{vv} のラベルを $f^*_{e_{vv}}$ とする．

各 $w \in \mathrm{succ}(v) \cap W$ につき，以下を行なう．

各 $u \in \mathrm{pred}(v) \cap W$ につき，代入 (v, u, w) を行う．

/* 辺の集合 $\{e \in \mathrm{in}(v) | h(e) \in W\}$ は E_U に，$\{e \in \mathrm{out}(v) | t(e) \in W\}$ は E_L に，それぞれ属するものとする．また，v にセルフループがあれば，それは E_L に属するのものとする．*/

[前進代入]

手続き

/* グラフ G_L に対して適用する．上の手続きにおける V の要素の処理順序を S とする．*/

$x \leftarrow b$

S の順に取り出した v につき，以下を行う．

もしセルフループ e_{vv} があれば，
$x_v \leftarrow f_{e_{vv}}(x_v)$
各 $w \in \mathrm{succ}(v)$ につき，
$x_w \leftarrow x_w \wedge f_{wv}(x_v)$

4.2.5 ガウス消去法の効率について

以下の性質は，実用上重要である．まず，準備として次の補題を証明する．

補題 4.1 L が完備束で $f \in F$ が連続性を持つとき，

$$f(1) = 1.$$

証明 関数の連続性の定義と，$\inf(\emptyset) = 1$ が成り立つことを用いて，

$$\begin{aligned} f(\inf(\emptyset)) &= \inf(\{f(x) | x \in \emptyset\}) \\ &= \inf(\emptyset) \\ &= 1 \end{aligned}$$

（証明終り）

このように束の完備性と関数の連続性がいえるときには，b の要素が，式 (1.3) のように1要素を除いてすべて L の最大元1に等しい場合（これは1頂点からの最短路問題や，出発点のある通常のデータフロー問題に対応する），$b_s \neq 1$ なる出発点 s をかなめの順序 S の最後に置けば，前進代入が不要になる．なぜなら，前進代入のアルゴリズムの中で，$x_v = 1$ とした時，$f_{e_{vv}}(1) = 1$ および $f_{wv}(1) = 1$ が成り立つので，x_v と x_w の値に変化がない．

たとえば最短路問題の場合には，前進代入のアルゴリズム中の $f_{e_{vv}} = \iota$ であり，また f_{wv} は $f_{wv}(x) = x + d_{wv}$ という形の関数であるから（d_{wv} は頂点 v から w までの距離を表す），いずれにせよ $f(\infty) = \infty$ となり，前進代入の必要がないわけである．

この性質は，全頂点間の経路を解析する場合にも役立つ．このときは，V のすべての要素ごとに，それを出発点とする b を作って，その n 個の b につきそれぞれ前進代入と後退代入を実行して解を得る．出発点 s はかなめの順序 S の中で任意の位置にくることになるが，上の前進代入は，アルゴリズム中に現れる関数 f が，$f(1) = 1$ を満たす限り，S の先頭から出発点 s までの間は不要になる．すなわち，前進代入における関数評価（および $\wedge$ 演算）は，たかだか $n^3/6$ のオーダですむ．後藤 [68] は，とくに最短路問題についてここで述べたガウス型のアルゴリズムを提案し，全頂点間の最短路問題に対して，$n^3/6$ オーダの手間であることを指摘している．

4.3 逐次法と消去法の関係

消去法における基本操作の内，代入操作は逐次法における基本的な操作単位と一致する．したがって，消去法が逐次法と本質的に違う点は，還元操作の存在と，代入操作および還元操作を実行する順序をあらかじめスケジューリングしているという，2点である．

還元操作に相当する処理は，逐次法では閉路に沿った代入操作の繰り返しとして実現される．4.2.1 節で述べたように，この繰り返しの回数は問題の次数で定まる．特殊な場合として，最短路のような次数0の問題では，この繰り返しを行なう必要がない．いずれにせよ，還元操作も代入操作に帰着させるとすれば，消去法のアルゴリズムは，逐次法の処理の順序を，第3章に示したようなデータ構造 S の性質（スタック，待ち行列，優先順位つき待ち行列，など）に依存して決めるのではなく，事前に定めたスケジューリングに従うようにするものと解釈することができる．逐次法の処理順序をあらかじめ定めるという考え方の例には，すでにアルゴリズム I3 としてあげた Hecht & Ullman[21] のほか，Kennedy によるもの [31] などがある．また，Tarjan[56] も，消去法アルゴリズムを，Kennedy の着想を一般化した頂点の処理順序という観点から，統一的に述べている．

一般に，手間の上限を見積もれば，消去法の方が逐次法より効率がよい．しかし，逐次法には，実現が簡単であるという特徴のほか，グラフの構造をあらかじめ明示的に保持せずに，処理過程で必要に応じ頂点や辺を生成することも可能であるという便利な性質がある．探索アルゴリズムが，通常は逐次型の構造をしているのは，このためである．

消去法に対しても，消去する頂点の順序（行列表現でいえば，行列の行（列）の並べ順）に関しては任意性があり，その順序によって効率が左右される．この問題に関しては，次章で扱う．

第 5 章

構造の分解と消去順序

第4章のアルゴリズムでは，かなめの選択順序は，任意に1つを固定するものとした．しかし，この順序は効率に影響する．このような消去順序を定める問題は，構造解析などにおける大規模方程式の解法や，大規模な構造を持つ線形計画法の分野では，疎行列の分解と関連してよく研究されている．また，ネットワーク・アルゴリズムでも，グラフの構造と関連づけた多くの研究がある．本章では，われわれの定式化に従ってこの問題を整理するとともに，とくにデータフロー解析の消去型解法で研究されている，グラフ変形の順序と関連した消去順序を，行列の掃き出し順序との対応により考察する．さらに，新たな消去順序を提案し，その評価を行なう．

5.1 手間の評価基準

まず，アルゴリズムの手間の評価基準を考える．ここでは，アルゴリズム G3 および G4 をとくに対象とする．

基本となる演算は代入および還元であるが，これをさらに分解すると次の演算に帰着する．

1. 関数に関する演算
   (a) 関数の合成
   (b) 関数の交わり
   (c) 不動点関数の生成

(d) 関数の適用

2. グラフ操作

(a) 辺の生成

(b) 辺の削除

1回の代入操作には，1回の関数合成と1回の交わり演算または辺の生成が伴う．1回の還元操作には，1回の不動点関数生成と1回の関数合成が伴う．ただし，1つの頂点の先行点（または後続点）の集合に対し，不動点関数の生成は共通に1回だけ行なえばよい．

第4章では，アルゴリズム G3 と G4 を，取り出した頂点の集合を W というデータ構造に保存する形で記述した．以降の考察では，V から取り出す頂点の順序が意味を持ってくるので，あらかじめ適当な順序 S が定められているとして，アルゴリズム G4 を以下のように書き換える．ここで，頂点 $v \in V$ に対し S による順位 $(1, 2, \cdots, n(=|V|))$ を与える関数を $\mathrm{ord}(v, S)$ とする．文脈から想定している頂点順序 S が明らかな時は，S を省略して $\mathrm{ord}(v)$ と書くこともある．また，S の順序で頂点 v より後にある頂点の集合を $S^+(v)$，v より前にある頂点の集合を $S^-(v)$ と書く．すなわち，

$$S^+(v) = \{w \in V | \mathrm{ord}(w, S) > \mathrm{ord}(v, S)\},$$
$$S^-(v) = \{w \in V | \mathrm{ord}(w, S) < \mathrm{ord}(v, S)\}.$$

[アルゴリズム G4'　ガウス消去法（頂点順序指定）]

入力　アルゴリズム G2 と同様のグラフ $G = (V, E)$，および，適当な頂点の順序 S.

出力　グラフ $G' = (V, E_U \cup E_L)$．ここで，E_U は $\mathrm{ord}(v) > \mathrm{ord}(w)$ であるような頂点 v, w 間の辺の集合．E_L は，$\mathrm{ord}(v) \leq \mathrm{ord}(w)$ であるような頂点 v, w 間の辺の集合．順序 S の最後の頂点を t とすると，部分グラフ $G_U = (V, E_U)$ は，t に入る辺がない無閉路グラフ，部分グラフ $G_L = (V, E_L)$ は，辺の向きをすべて逆向きにしたときに，セルフループを除いて，やはり t に入る辺がない無閉路グラフとなる．

任意の定数ベクトル b に対して，方程式の解は，前進代入を部分グラフ G_L に対して行い，その後，後退代入を部分グラフ G_U に対して行ったものとなる．

手続き

S の順に取り出した $v \in V$ につき，以下を行なう．

もし v にセルフループ e_{vv} があれば，

各 $u \in \mathrm{pred}(v) \cap S^+(v)$ につき，前還元 (v, u) を行う．

e_{vv} のラベルを $f^*_{e_{vv}}$ とする．

各 $w \in \mathrm{succ}(v) \cap S^+(u)$ につき，以下を行なう．

各 $u \in \mathrm{pred}(v) \cap S^+(v)$ につき，代入 (v, u, w) を行う．

アルゴリズム G3 を G4 と比べると，本質的な違いは辺を削除する点だけであるが，順序 S を考慮すれば，辺の削除操作は陽に実行する必要がない．$\mathrm{ord}(v) > \mathrm{ord}(w)$ であるような頂点 v, w 間の辺を，無視すればよいからである．

このアルゴリズムの手間を評価する．

1. 不動点関数生成は高々 $|V|$ 回（正確には，G_L におけるセルフループの数）．
2. 関数合成は $\sum_{v \in V} |\mathrm{in}(v, G_U)| \times |\mathrm{out}(v, G_L)|$ 回．
3. 関数の交わり演算と辺の生成操作の合計数は，$\sum_{v \in V} |\mathrm{in}(v, G_U)| \times |\mathrm{out}(v, G'_L)|$ 回．

ここで，in と out は対象とするグラフを第2引数で明示するように記法を変更している．また，グラフ G'_L は G_L からセルフループを除いたものである．

前進代入ではグラフ G_L の辺の数の関数適用と交わり演算を行なう．後退代入では，グラフ G_U の辺の数の関数適用と，辺の数から頂点の数を引いた回数の交わり演算とを行なう．

以上の手間の評価は，アルゴリズムの実行の結果できるグラフ G_U と G_L の辺の数などの特性に基づいているため，実行前の評価としては使えない点に注意がいる．グラフ G_U および G_L の辺の数は，アルゴリズムの実行によって一般に増加する．その増加量が効率に影響する．

5.2 ブロック三角化

第4章の消去型アルゴリズムの導出においては，行列表現を出発点としながらも，グラフ表現を，ここで定式化している問題に対する自然な表現方法として，強調してきた．しかし，頂点の順序を固定した場合には，行列の行と列の並びが自然な意味を持ってくる．また，線形方程式系の解法で消去順序はよく研究されているので，その対応を見るためにも以下では行列表現を併用していく．

順序 S を定めて，それにしたがった行と列の順序を与えた行列表現を作った時，行列が上三角化されていれば，すなわち $i > j$ であるような行列の (i,j) 要素はすべて最大元 1 の定数関数である τ（線形方程式系の係数 0 に対応）であれば，消去演算は明らかに不要になる．この場合，対応するグラフが無閉路グラフであることも，ほぼ明らかである．

[例] 無閉路グラフと対応する行列の例を，図に示す．行列中の $\times$ は τ 以外のラベルを持つ辺の存在を示す．対角要素は $\circ$ で表す．

	1	2	3	4	5
1	$\circ$	$\times$	$\times$		
2		$\circ$	$\times$		$\times$
3			$\circ$	$\times$	$\times$
4				$\circ$	$\times$
5					$\circ$

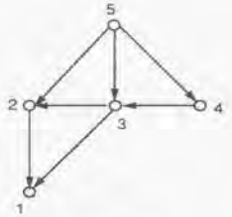


図 5.1 無閉路グラフと上三角行列

なお，以下では図5.1の行列で $\times$ で表したような，τ 以外のラベルを持つ辺に対応する行列要素を，単に要素と呼ぶ．

同じ無閉路グラフに対しても，頂点の順序によって，対応する行列が上三角行列になるとは限らない．上三角行列を与える順序を得るには，無閉路グラフが作る半順序関係，すなわち，頂点 v から w に向かう辺があれば $v \prec w$ とするという関係に対して，位相整列した順序の逆順とすればよい．

このような行（列）の順序を定めて行列を三角化する手法をさらに拡張した，ブロック三角化の方法もよく知られており，それに基づいたかなめの順序で掃き出し操作を行うことにより，（一般に非線形を含む）方程式系の解法の効率を向上させることは，実際よく行なわれる [13, 22, 75]．筆者も，1976年に数理計画法のシステムを開発した際，線形計画法における基底行列の逆行列を求めるのにこの手法を採用し，それまでの方法と比べて，行列の逆転後の非零要素を格段に減少させることができた経験を持つ [72]．その時点で，文献 [22] の方法は知られていたが，商用の数理計画システムにそれを実現した例は，まだなかったと思われる．

行列のブロック三角化を形式的に定義すると，次のようになる（[64, 72] 参照）．

行列 A の行番号の集合を I，列番号の集合を J としたとき，I の部分集合 I_i と J の部分集合 J_i の対の列 $((I_1J_1)(I_2J_2)\cdots(I_lJ_l))$ を求めること．

ただし，次の条件を満たす．

1. $I_i(i=1,\cdots,l)$ は I の分割をなし，$J_i(i=1,\cdots,l)$ は J の分割をなす．また I_i と J_i の位数は等しい．
2. I_i の各行は，$\bigcup_{k=1}^{i-1} J_k$ の任意の列に要素を持たない．（したがって，J_i の各列は，$\bigcup_{k=i+1}^{l} I_k$ の任意の行に要素を持たない．）
3. I の分割 $I_i(i=1,\cdots,l)$，および J の分割 $J_i(i=1,\cdots,l)$ は，条件 (1)(2) を満たすもののうちで分割の細かさが極小である．（すなわち，任意の I_i，J_i をさらに分割して，(1)(2) を満たすような列を作ることはできない．）

[13], [22], [75] や [72] の方法は，行と列とを独立に並べ換えてブロック三角化するものである．その基本的な方法は次のように書ける．

1. 行と列とのマッチングを求める．マッチングとは，行と列との1対1写像で，写像関係にある行 i 列 j に要素が存在するものをいう．
2. 列に対応する頂点からなる集合 V_C と，行に対応する頂点からなる集合 V_R との間の2部グラフを作る．行 i 列 j に要素があれば j から i への辺

を作る．さらに，i と j がマッチング関係にあれば，行 i から列 j への逆向きの辺を作る．

3. 生成した2部グラフを強連結成分に分解する．各強連結成分に含まれる行と列を対にして，分割 (I_i, J_i) を作る．それらを，各強連結成分を1つの頂点に縮約して得られる無閉路グラフで定まる半順序関係に関して，位相整列したものの逆順に並べる．

われわれの定式化では行（等式）に列（変数）が本来的に対応しているから，行と列の対応関係を崩さずに並べ換えなければならない．この場合はさらに簡単で，もとのグラフをそのまま強連結成分分解すればよい．すなわち，もとのグラフがすでに強連結である場合を除けば，同じ強連結成分に属する頂点はまとめて連続するようにし，強連結成分間は，それらの到達可能性に基づく半順序関係を保つように位相整列させた頂点順序の逆順をとると，かなり効率の良い消去を実現できる．

なお，強連結成分分解に関しては，Tarjan[50] に深さ優先探索に基づく $O(m+n)$（m, n はそれぞれ辺と頂点の数）の手間の効率の良いアルゴリズムがある．

5.3 簡約可能グラフ

5.3.1 簡約可能グラフの定義

すでに述べたように，アルゴリズム G3 や G4 における効率を決定する尺度として，代入操作によって新たに生成される辺の数を用いることができる．この生成される辺とは，線形方程式系における疎行列の扱いの中では，非零要素の混入 (fill-in) と呼ばれる概念に対応する．非零要素の混入を最小にするような消去順序を求める問題は，一般に NP 完全であることが知られている [40, 61]．

データフロー解析においては，消去型の解法に関連して，特定の性質を持つ簡約可能グラフ (reducible graph) がよく研究されている．既存のほとんどの消去型データフロー・アルゴリズムは，簡約可能グラフに対してすぐれた効率をあげるものである．以下で，簡約可能グラフに対する既存のアルゴリズムを，消去順序と辺の生成という観点から考察し，その特性を明らかにする．

簡約可能グラフは，通常，フローグラフに対して定義される．プログラムの構造を表現したフローグラフの意味的な定義については，1.2.3 節で与えたが，グラフとしては，有向グラフで，出発点と呼ぶある特定の頂点から，他の任意の頂点に到達可能なものをフローグラフと定義することにする．すなわち，フローグラフは，頂点集合 V，辺集合 E，出発点 $s \in V$，の3つ組 $G = (V, E, s)$ で定義される．これまでの議論では，非連結なグラフも対象としてきたが，第1章で例を示したようなほとんどの解析は，連結成分ごとに別々に行なえばよいので，出発点と他の頂点との連結性の仮定は，一般性を損なうものではない．なお，1つの頂点を出発点として特別視することの意味については，後に改めて考察する (5.5 節)．

簡約可能グラフの定義の仕方はいろいろあるが，1つの定義は次のようである [19]．

フローグラフ $G = (V, E, s)$ が簡約可能とは，次の2つのグラフ変形の繰り返しによって，頂点 s のみからなり辺をもたないグラフに簡約されるものを言う．

T1: e が $h(e) = t(e)$ であるようなセルフループであれば，それを取り除く．

T2: $w \neq s$ であるような頂点 w に対して，すべての辺 $e \in \mathrm{in}(w)$ が同一の頂点 v を始点とするとき，$\mathrm{in}(w)$ にあるすべての辺を削除し，$\mathrm{out}(w)$ にあるすべての辺の始点を v に取り替え，それによって頂点 w を v に吸収する．

T1, T2 変形を続けて，もうそれ以上変形ができない時に得られるグラフは，T1，T2 の適用順序によらないことが知られている [19]．したがって上の簡約可能性の定義は，妥当である．

最初に与えられたグラフに対しても，T2 による変形の途中でも，ある頂点から別の頂点への辺は高々1つであるように保つ（すなわち，ある頂点対に2つ以上の辺があったら，ただちにそれを1つの辺となるように合併する）ものと仮定すると，T2 操作は次のように言い換えてもよい．

T2': $w \neq s$ であるような頂点 w に入る辺がただ1つのとき，その辺 e を削除し，$\mathrm{out}(w)$ にあるすべての辺の始点を v に取り替え，それによって頂点 w を v に吸収する．

以下では，必要な合併操作は常に行なわれるものと仮定し，T2 操作を T2' の形で考える．

T2 操作の条件により，同じグラフでも出発点のとり方によって簡約可能になったり簡約不可能になったりすることに注意しよう．

[例] 次のグラフでは，出発点を 1, 4, 7 のいずれかにとると簡約可能となるが，それ以外の頂点を出発点とすると簡約可能でない．

	1	2	3	4	5	6	7
1	○					×	×
2	×	○			×		
3		×	○				
4	×			○	×		
5			×		○		
6		×				○	
7				×			○

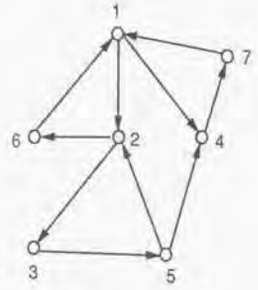


図 5.2 出発点のとり方に依存する簡約可能グラフ

5.3.2 簡約順序

簡約可能グラフを対象としたデータフロー解析で，計算量を抑える工夫を凝らした消去法アルゴリズムの代表として，Ullman[58], Graham & Wegman[17], Tarjan[56] の 3 つをとりあげ，考察を行なう．いずれの方法も，消去順序は本質的に T1, T2 変形によってグラフを簡約する過程に基づいている．すなわち，T2 変形で吸収される頂点の順序を，そのまま消去順序とする．

もともと，T1 が還元操作に，T2 が代入操作に関連することは，明らかである．ここで，その対応を具体的に見るために，T2 によって吸収削除される頂点の順序を簡約順序と呼び，それにしたがって行列を並べ変えたとしよう．図 5.2のグラフで頂点 1 を出発点とした時，簡約順序（の 1 つ）は，(7, 6, 5, 3, 2, 4, 1) となる．それにしたがって並べ替えた行列の構造は，次のようである．

	7	6	5	3	2	4	1
7	○					×	
6		○			×		
5			○	×			
3				○	×		
2			×		○		×
4			×			○	×
1	×	×					○

この頂点順序に応じて簡約を行なう．すなわち，この順序で取り出した頂点 v につき，もし v から出て v に入るセルフループがあればそれを T1 操作で取り除いた上で，T2 操作により v をしかるべき頂点（簡約順序で v より後にあるはずである）に吸収する．

なお，簡約順序を効率良く求める方法としては，Tarjan[51] がある．

各頂点 v に対し，簡約によりそれが吸収される先の頂点をヘッダと呼び，header(v) と書く．もとのグラフの頂点集合と，header(v) から v へ向かう辺によって作られるグラフを考えると，それは明らかに出発点 s を根とする木になる．この木をヘッダ木という [51]．上の例題に対するヘッダ木は，図 5.3のようになる．なお，T1 操作が行なわれるのは，頂点 2 と 1 である．

図 5.4では，T2 によって生成された辺を √ で示している．ただし，対角上にできる辺，すなわちセルフループは ⊗ で表す．

この例からも，簡約過程がガウス消去法（アルゴリズム G3 または G4）にちょうど対応していることが分かる．この対応に関し，次の性質がいえる．

補題 5.1

1. 簡約順序に対応した行列表現では，初期行列の各行の対角要素より右には高々 *1* つの要素しか存在しない．すなわち，初期行列の上三角行列に対応するグラフは，森（木の集合）になる．

2. *T1* および *T2* による変形過程では，次に *T2* を施どこすべき行（グラフの縮小に応じて行列を縮小させていくとすれば，その時点の行列の第 *1* 行）の対

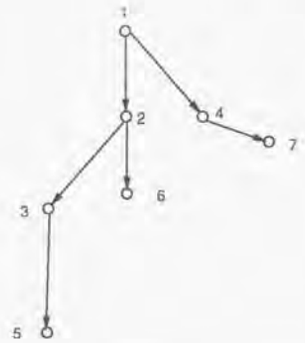


図 5.3 ヘッダ木

角要素より右には，ちょうど *1* つの要素があり，他の行の対角より右には高々 *1* つの要素がある．したがって，*T2* 操作による辺の生成（消去法において，あるかなめによる *1* つの行の掃き出しで生ずる要素の混入に対応）は，高々 *1* つである．

3. 最終的に得られる行列全体の上三角行列に対応する部分グラフは，ヘッダ木を構成する．

証明 いずれもこれまでに述べた簡約過程の性質から，明らかである．（証明終り）

上の例では，初期行列の上三角行列がそのままヘッダ木に対応し，簡約の過程で森が合成されて木になるという変化が生じていないが，一般には木同士を結合してより大きな木とするような辺の生成が，起こりうる．

5.3.3 簡約順序に基づく消去法

簡約順序で単純に G4' アルゴリズムを実行しても，効率は必ずしもよくない．上の補題の性質 2 と，5.1 節に示した手間の評価から，関数合成は $\sum_{v\in V}|\mathrm{out}(v,G_L)|$ 回，関数の交わり演算と辺の生成操作の合計数は $\sum_{v\in V}|\mathrm{out}(v,G_{L'})|$ 回である

	7	6	5	3	2	4	1
7	∘					×	
6		∘			×		
5			∘	×			
3				∘	×		
2			×	√	⊗		×
4			×	√	√	∘	×
1	×	×			√	√	⊗

図 5.4 簡約後の行列表現

ことがいえる．すなわち，いずれもグラフ G_L の辺の数に比例した手間となる．しかし，もとのグラフが疎であっても，辺の生成により G_L は最悪の場合 $O(n^2)$ の辺を持つので，手間のオーダは $O(n^2)$ となる．

簡約順序に基づく消去法では，この手間を $O(m\log n)$（ただし $m=|E|$）に押える工夫をしている．

それを述べるための準備として，消去法の基本的なアルゴリズムを変形する．アルゴリズム G4' では，ある順序で取り出した頂点 v をかなめとし，v より順位の大きいすべての後続点 w と，v より順位の大きいすべての先行点 u の組合せに対して，代入 (v,u,w) を実行した．これは行列の上の操作イメージでいえば，かなめ v の列の v より下側にある要素を，v の行で掃き出していることになる．この順序を変えて，v の行の左側にある要素を，左から順に v より上の対応する列に対角要素を持つ行で掃き出していってもよい．この掃き出しの順序を図示したのが，図 5.5である．

これを，アルゴリズムとして書くと次のようになる．

[アルゴリズム G4" ガウス消去法 (横方向掃き出し)]

入力，出力は G4' を参照．

手続き

S の順に取り出した $v\in V$ につき，以下を行なう．

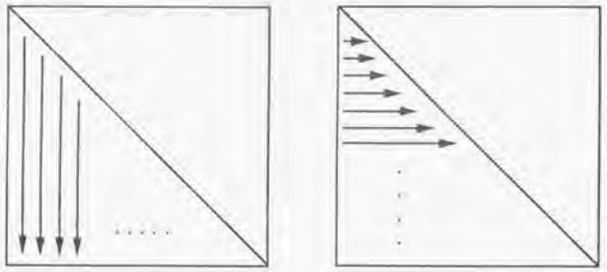


図 5.5 掃き出し順序 — 左が G4', 右が G4" に対応

各 $w \in \mathrm{pred}(v) \cap S^-(v)$ につき，S の順序で下位にあるものから順に以下を行なう．

　各 $u \in \mathrm{pred}(w) \cap S^+(w)$ につき，代入 (w,u,v) を行う．

もし v にセルフループ e_{vv} があれば，

　各 $u \in \mathrm{pred}(v) \cap S^+(v)$ につき，前還元 (v,u) を行う．

　e_{vv} のラベルを $f^*_{e_{vv}}$ とする．

ここで消去順序の変更にともない，各頂点 v に対する還元操作と代入操作の順序が，入れ替わっていることに注意．なお，とくに簡約順序の場合には，この消去法に対応するグラフ変形は，補題 5.1 と同様の理由で，上三角部分に相当する部分グラフが，ヘッダ木を構成するものとなっている（後述の補題 5.2）．したがって，手続きの 4 行目の $u \in \mathrm{pred}(w) \cap S^+(w)$，および 6 行目の $u \in \mathrm{pred}(v) \cap S^+(v)$ は，それぞれちょうど 1 つしか存在しない．

以降の考察は，この手続きで，1 つの頂点 v を処理する時点に固定して考える．この時点での行列の上三角部分に相当するグラフを G_U，下三角部分に相当するグラフを G_L とする．

上に述べたように，v と w を定めると u はただ 1 つに定まる．これら 3 点の間に，次の関係が成立する．

補題 5.2 簡約順序によるアルゴリズム *G4"* で，上に述べた 3 点 v, w, u に関し，

1. グラフ G_U 上で，頂点集合 $S^-(v)$ と，それに属する頂点のヘッダからなる部分グラフは，それらの頂点を含むヘッダ木の部分グラフ（森）となっている．

2. もし $\mathrm{ord}(u) > \mathrm{ord}(v)$ ならば，$u = \mathrm{header}(v)$ であり，$\mathrm{ord}(u) \leq \mathrm{ord}(v)$ なら，ヘッダ木上に $\mathrm{header}(v)$ から u に至る（さらに辺 (u,w) により w に至る）経路が存在する（図 *5.6* および *5.7* 参照）．

証明　1. を，v の順序に関して帰納的に仮定する（帰納法の基底が成り立つことは自明）．$w \in S^-(v)$ だから，$u = \mathrm{header}(w)$ である．T2 操作により w を u に吸収した時，辺 (u,v) ができるが，$\mathrm{ord}(u) > \mathrm{ord}(v)$ なら，u より先に v が吸収されることになるので，その吸収先は u でなければならない．すなわち，$\mathrm{header}(v) = u$．$\mathrm{ord}(u) \leq \mathrm{ord}(v)$ の場合は，$u = v$ であるか，$u \in S^-(v)$ である．後者の場合は，上の議論の w を u に取り替えることを必要に応じて繰り返すことにより，最終的に u は，やはり $\mathrm{header}(v)$ に行きつくか v に行きつくことになる．いずれにせよ，ヘッダ木上で $\mathrm{header}(v)$ から u（そして w）に至る経路がある．すなわち，2. が証明できた．

1. の帰納法を進めるために，$v \neq s$ と仮定する．$\mathrm{header}(v)$ から v への辺が初めから存在している場合は，v の処理が終った時点で，$S^-(v) \cup \{v\}$ に対しヘッダ木の部分グラフができていることは明らかである．そうでない場合，$w \in \mathrm{pred}(v) \cap S^-(v)$ であるような w のうち（代入操作によって新たに作られた辺によって $\mathrm{pred}(v)$ が途中で増えることも考慮），少なくとも 1 つは，$u = \mathrm{header}(w)$ で，$\mathrm{ord}(u) > \mathrm{ord}(v)$，すなわち $u = \mathrm{header}(v)$ となるものがある．そうでないと，$S^-(v)$ のすべての頂点に T2 操作を行なった後，v に入る辺が存在しないことになり，矛盾である．このような v,w,u に対し代入 (w,u,v) を行なえば，$u = \mathrm{header}(v)$ から v への辺が生成され，やはり $S^-(v) \cup \{v\}$ に対しヘッダ木の部分グラフができることになる．（証明終り）

なお，この補題の後半は Tarjan[56] の lemma 3 と実質的に同じである．

補題 5.2から，アルゴリズム G4" では，$w \in \mathrm{pred}(v) \cap S^-(v)$ に対して，ヘッダ木上に v から w または $\mathrm{header}(v)$ から w に至る経路があることが分か

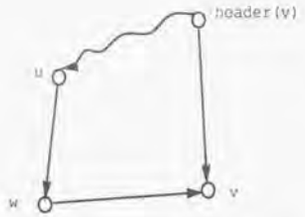


図 5.6 消去法における 3 点の関係

る（図 5.7参照）.

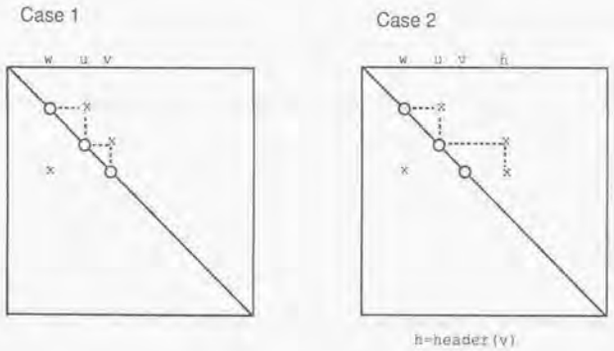


図 5.7 ヘッダ木上の経路

この経路は，辺 $e=(w,v)$ とその時点で作られているヘッダ木の部分グラフ（森）で定まる．この経路 p_e に対応する頂点列を $u_0,u_1,\ldots,u_k$ とする．ここで，$u_0=v$ または $\mathrm{header}(v)$，$u_k=w$ である．$u_0=v$ の場合（図 5.7のケース 1），p_e に e を加えると閉路となる．$u_0=\mathrm{header}(v)$ の場合（ケース 2）は，閉路とならない（Tarjan[51] では，前者のような辺を $\mathrm{cycle}(v)$，後者を $\mathrm{non\text{-}cycle}(v)$ と分類している．これはまた，Hecht & Ullman[20] における後退辺と交差辺に，ほぼ対応する概念である）.

辺 e の消去操作，すなわち代入 (w,u_{k-1},v) によって，新たな辺 (u_{k-1},v) が生成される[1]．さらに，(u_{k-1},v) を消去するという操作を続けて，k 回の代入操作により，e および経路 p_e 上の辺によって新たに生成される辺が消去される．その結果，ケース 1 では v にセルフループが，ケース 2 では辺 $(\mathrm{header}(v),v)$ が生成される．後者は，ヘッダ木の一部となる．

ここで，経路 p_e の長さ $\mathrm{len}(p_e)$ を，そこに含まれる辺の数で定義する．すなわち，$\mathrm{len}(p_e)=k$．そうすると，この一連の操作における手間は，$\mathrm{len}(p_e)$ 回の代入操作（関数合成と辺の生成各 1 回）となるから，全体では，$\sum_{e\in E_L}\mathrm{len}(p_e)$ に比例する手間となる．ただし，ここでは E_L は最初に与えられたグラフに対応する行列の下三角部分に存在する辺，すなわち $E_L=\{e\in E|\mathrm{ord}(h(e))<\mathrm{ord}(t(e))\}$ とする．

このままでは，最悪の場合の手間はまだ $O(n^2)$ である．しかし，1 つの辺 e に対する $\mathrm{len}(p_e)$ の上限は，その時点で $w=h(e)$ が属する部分木の高さで与えられることに注意しよう[2]．そこで，消去の過程でこれら部分木の高さを低くするような変形を適当なタイミングで行なえば，全体の手間を本質的に減らすことが期待できる．このアイディアが，Ullman[58], Graham & Wegman[17], Tarjan[56] の 3 手法に共通するものといえる．たとえば木を平衡 2 分木にすれば，その高さは $\log_2 n$ となる．平衡 2 分木を保つ手間も $\log_2 n$ のオーダですむ．したがって，全体の手間は $m\log_2 n$ でおさえられる．

具体的には，次のような方法を取ることができる．辺 e に対して定まる経路 p_e（頂点列 $u_0,u_1,\ldots,u_k$）に対して，消去の順序を変えて，$(u_1,u_0,u_2),(u_2,u_1,u_3),\ldots$ の順にそれぞれを引数とする代入を実行し，最後に e の消去（代入 (u_k,u_{k-1},v)）を行なう．いま，$S^-(v)$ の下三角部分はすでに掃き出され辺がないことを仮定しているから，このような消去順序によっても，下三角部分については影響を受けない．上三角部分については，消去が起こる．その結果，図 5.8のように，

[1] すでに辺 (u_{k-1},v) が存在する場合は，新たな辺を生成する代わりに，すでにある辺に合併させればよいが，ここではこれまでの仮定と異なり，辺が同じ頂点対の間に重複しても別に生成させるものと仮定する．

[2] 一般に経路 $e_1,e_2,\cdots,e_k$ の長さを，含まれる辺の数 k として，木の高さを根から任意の頂点に至る経路の長さの最大値で定義する．

木の高さが縮小される．この操作は，上三角部分も掃き出すという意味では，ガウス・ジョルダン法に相当する．しかし，上三角部分の構造を森に保っているために，手間が省略されている．実際，この消去（代入）操作と，アルゴリズム G4" における下三角部分の掃き出しに相当する操作とを比べると，1 つの辺 e に対して，どちらも同じ $\mathrm{len}(p_e)$ 回の手間となっている．

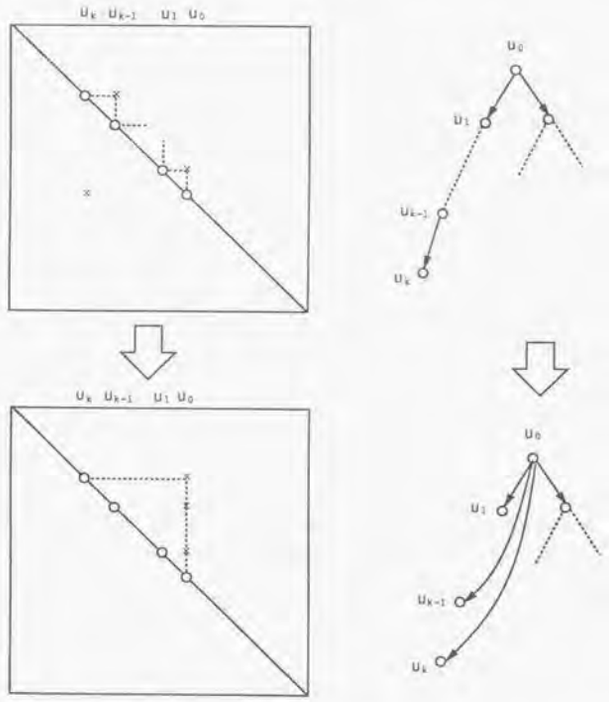


図 5.8 木の高さの縮小

以上述べた方法は，説明の仕方は異なるが，本質的に Tarjan[56] に基づいている．Tarjan[54] により，この手間が $O(m \log n)$ であることが分かる．

以上をまとめて，アルゴリズムとして記述する．

[アルゴリズム G5　簡約順序による消去法]

入力　アルゴリズム G2 と同様な意味で，方程式 $x = Ax \wedge b$ に対応するフローグラフ表現 $G = (V, E, s)$．ただし，ベクトル b は出発点 s にのみ 1 でない要素 b_s をもつものとする．すなわち方程式は式 (1.3) の形で考える．このフローグラフは簡約可能とし，簡約順序 S が与えられているものとする．

出力　フローグラフ $G_U = (V, E_U, s)$．ただし，G_U は s を根とする木である．方程式を満たす解 x_v は，木上で s から v に至る経路 $e_1, e_2, \ldots, e_k$ に対して，$f_{e_k} \cdots f_{e_2} \cdot f_{e_1}(b_s)$ によって求められる．

手続き

S の順に取り出した $v \in V$ につき，以下を行なう．

S の順に取り出した $w \in \mathrm{pred}(v) \cap S^-(v)$ につき，以下を行なう．

G_U 上で w の属する部分木の根から w に至る経路 p を求め，圧縮 (p) を行なう．

もし v にセルフループ e_{vv} があれば，

v の G_U 上の親（$\mathrm{pred}(v)$ の唯一つの要素）u に対し，前還元 (v, u) を行なう．

圧縮 (p)

入力　辺 $e = (w, v)$ に対応する，部分木の根 r から頂点 w に至る経路 p．その頂点列を $u_0, u_1, \ldots, u_k$ とする（$u_0 = r, u_k = w$）．さらに，$u_{k+1} = v$ とする．

出力　辺 e を消去し，それに伴い部分木の高さを低くするような圧縮変形を行なったグラフ G．

手続き

i を 1 から k まで動かして，代入 (u_i, u_{i-1}, u_{i+1}) を行なう．

このアルゴリズムについて，以下の点を注意する．

1. 出発点と初期値

ここでは，これまでに扱ってきた消去法アルゴリズムが方程式 (1.2) の形の一般的な定数ベクトル b を想定していたのと異なり，出発点にのみ 1 でない要素を持つ方程式 (1.3) を仮定している．これは対象とするグラフが出発点 s をもつフローグラフであることからも，自然である．とくに簡約可能グラフを対象としているので，アルゴリズム G1 のように，一般のベクトル b に対して s から b に要素をもつ頂点に辺をもうけるという拡張を行なうと，簡約可能性が必ずしもいえなくなる点に注意がいる．

この b に対する仮定から，初期値 b_s を変えても，前進代入に相当する計算は通常不要である（4.2.5 節にある議論を参照）．また，このアルゴリズムではガウス・ジョルダン法のように上三角部分も一部消去しているが，同じ理由によってベクトル b に対して対応する演算をほどこす必要がない．

2. 出力グラフ G_U と後退代入

結果としてできるグラフ G_U は木であるから，任意の頂点 v に対する解 x_v を求めるには，s から v への経路に沿って初期値 b_s を変換していけばよい．しかし，全頂点についての解を求めるには，順序 S の逆順にアルゴリズム G3 における後退代入に相当する計算をした法が，効率が良い．

最終的な木 G_U の高さは，多くの場合 1 になる．1 より大きい場合でも，さらに木の高さを縮める変形，すなわち根からの経路の長さが 2 以上の頂点に対して圧縮を行なえば，木の高さを 1 にできる．この変形を行なえば，上記の解を求める計算（たとえば後退代入）は，実は不要になる．しかも，高さ 1 にまで木を変形することに要する手間の上限は $O(m \log n)$ で変わらない．この変形結果は，アルゴリズム G1 のガウス・ジョルダン法で最終的に出力されるグラフに対応する．

ここで述べたアルゴリズムは Tarjan[56] にあるものとほぼ同じであるが，Graham & Wegman[17] と Ullman[58] との異同に触れておく．基本的にグラフを木に変形して計算の構造を捉えようとする考え方は，共通している．すなわち，

手続きの途中の時点で部分的に構成されている木（または森）は，根から頂点に至る経路上の関数を合成することによって，その頂点に対する部分的な解を与えるという性質を保っている．同じ性質を持つ木でも，その構造により計算の効率に違いが生じる．そこで，効率の良い木に変形していく工夫をしているのも共通だが，その工夫の仕方に若干の違いがある．

Graham & Wegman は，木としてヘッダ木を取り扱うのではなく，s を根とする 1 つの極大木を最初に固定する．簡約順序にしたがえば，T1, T2 の変形の過程に，この木の高さを圧縮する処理を $m \log n$ の手間で導入することができ，それによって T2 操作の手間も同じ上限を持つようにできることは，上に述べた方法とほとんど同じである．Graham & Wegman は，頂点の簡約順序という概念ではなくて，与えられたグラフのある性質を持った部分グラフである領域（後述）の順序という概念を利用しているが，実質的にはまったく同じである．

Ullman は，やはり簡約順序で T1, T2 変形を行ないながらヘッダ木に相当するものを構成していくが，効率の良い木として 2-3 木を導入するところが異なる．2-3 木の構成はアルゴリズムとしてやや複雑であるが，手間の上界は同じく $m \log n$ となる．

5.3.4 簡約可能グラフのさまざまな性質

前節のアルゴリズムは効率的に優れたものであるが，次のような問題点もある．

1. 計算の手間としては $O(m \log n)$ のオーダが実現されているが，データ構造としてたとえば森を扱い，その高さを圧縮する処理をするとか，2-3 木の構造を維持するというような，かなり複雑な処理を行なわなければならない．
2. 簡約可能グラフにのみ適用可能で，簡約不可能なものに対して自然な拡張ができない．

次節以降で，簡約不可能なグラフに対しても有効で，アルゴリズムも簡便なものを考察する．計算の手間の上限は $O(n^3)$ となるが，疎な性質を持つグラフに対しては平均的によい効率を実現するものである．

その準備として本節では，簡約可能グラフの持つ性質について，改めて整理しておく．もともと簡約可能グラフは，プログラムのフローグラフにおけるループの入れ子構造に応じて，データフローを解析する方法に関連して，提案された [3, 4, 5]．その後，Hecht & Ullman[19, 20] 等により，その性質が精緻化された．多くの性質があるが，とくに重要なものを，以下に証明なしに挙げる．

初めに，フローグラフのある頂点が別の頂点を“支配する (dominate)”という関係を定義しておく．フローグラフ $G = (V, E, s)$ の頂点 v が u を支配するとは，s から u への任意の経路上に v が含まれることをいう．

閉路への入口の一意性

性質 1 フローグラフは，図 5.9のような2つの入口を持つ閉路を部分グラフとして含まない時（言いかえると，任意の閉路に対して，その上のすべての頂点を支配する頂点が，閉路上にただ1つ存在する時），またその時に限り，簡約可能である．

図 5.9 簡約可能でない部分グラフ

極大無閉路部分グラフ

性質 2 フローグラフ G と同じ頂点集合 V をもつ G の部分グラフで，閉路を含まないという性質を満たす極大なグラフを極大無閉路部分グラフ（通称ダグ (dag; directed acyclic graph)）と呼ぶ．G が簡約可能であるのは，そのダグが唯一である時であり，またその時に限る．

ダグを得るには，まず出発点を根とする極大木を，たとえば深さ優先探索などによって求める（3.6.1 節参照）．極大木によって定まる頂点間の半順序を $\prec$ で表す．すなわち，辺 e が木に属する時 $h(e) \prec t(e)$ とし，その推移的閉包をとる．さらに，

$$v \preceq u \leftrightarrow v = u \,||\, v \prec u$$

とする．この時，木に属さない辺を次の3種類に分類できる．

1. 前進辺 (forward edge)： $h(e) \prec t(e)$ であるような辺 e．
2. 後退辺 (back edge)： $t(e) \preceq h(e)$ であるような辺 e．
3. 交差辺 (cross edge)： 前進辺でも後退辺でもない辺．

性質 3 簡約可能グラフのダグは，極大木の辺の集合に前進辺と交差辺をすべて付け加えることによって，構成できる．その際，極大木のとり方に依存しない．

性質 4 フローグラフに対して深さ優先極大木をつくったとする．その極大木で定まるすべての後退辺 (u, v) について，v が u を支配するとき，またそのときに限り，もとのフローグラフは簡約可能である．

領域

Ullman による領域の定義は次のようである [58]．フローグラフ $G = (V, E, s)$ に対し，フローグラフ $G_r = (V_r, E_r, s_r)$ が次の条件を満たすとき，G_r を s_r をヘッダとする G の領域という．

1. $V_r \subset V, E_r \subset E$ かつ $s_r \in V_r$．
2. s から任意の $v \in V_r$ に至る任意の経路は，必ず s_r を入口とする．すなわち，任意の経路 $e_1, e_2, \ldots, e_k (h(e_1) = s, t(e_k) = v)$ に対し，ある $i \in \{1, 2, \ldots, k\}$ が存在して，
   (a) $h(e_i) = s_r$．
   (b) $e_i, e_{i+1}, \ldots, e_k$ はすべて E_r に属する．

この領域は，フローグラフの簡約という概念に以下のように結びつけられる．フローグラフの T1, T2 による簡約過程で得られるグラフの各頂点は，もとのグラフの頂点と辺の部分集合を代表し，各辺もまたもとのグラフの辺の部分集合を代表していると考える．代表 (representation) の定義は次のようである．

1. もとのグラフでは，すべての頂点と辺はそれぞれ自分自身を代表する．
2. T1 が頂点 v のセルフループ e に適用された場合，その結果得られるグラフにおける v は，v と e が代表していたものの合併を代表する．
3. 頂点 v から w への辺 e に T2 が適用されて，e が消去された場合，その結果得られるグラフにおける v は，v と w と e が代表していたものの合併を代表する．またこの変形前に，辺 $e_1 = (v, u)$ と $e_2 = (w, u)$ とがあるような頂点 u が存在する場合は，変形後に e_1 は e_1 と e_2 が代表していたものの合併を代表する．

性質 5 簡約過程の任意の時点で，各頂点が代表するグラフは，もとのグラフの領域となっている [58].

Graham & Wegman[17] は，領域の概念にさらに制約を加えた “簡約集合 (reduce set)” を用いている．簡約集合のもとの定義は，極大木に対して定まる後退辺（上に述べたように，簡約可能グラフの場合，辺が後退辺かどうかは極大木の取り方によらない）の終点を h としたとき，その h によって定まる次の性質をもった頂点 v の集合である．

1. v は h に支配される．
2. v から h に至る経路で，その上のすべての頂点が h に支配されるものが存在する．

簡約集合によって定められる部分グラフは，明らかに h を入口とする領域となるが，さらに強連結であるという条件が加えられている．

Graham & Wegman は，これら簡約集合のうち，入口が他の簡約集合の入口を支配していないものから順に簡約し，それでさらに無閉路グラフが残ればそれを簡約するという方法を取っている．

5.3.5 ダグ順序

*ダグ*が定める頂点間の半順序関係に関して，位相整列した順序の逆順を*ダグ順序*と呼ぶことにする．簡約可能グラフの*ダグ順序*を求めるのには，深さ優先極大木を作りながら，その “後順 (postorder)” を生成するのが効率が良い [20].

*ダグ順序*を 1 つ選んで S とする．問題を，S の頂点順序に対応して行と列を並べた行列により表現すれば，明らかに，その上三角部分に*ダグ*に属する辺に対応する要素が現れ，下三角部分に*ダグ*に属さない後退辺（すなわち，閉路を作る辺）に対応する要素が現れる．G が簡約可能であれば，次の性質が満たされる．

定理 5.1 *ダグ*に基づく順序 S に従って並べた行列では，下三角部分に現れる要素の数（$f_{ij} \neq \tau, i > j$ であるような要素数）が最小となる．

証明 簡約可能グラフでは，*ダグ*がただ一つ定まる．したがって，*ダグ*の辺に対応する上三角部分の要素数は極大であると同時に最大となり，それに応じて下三角部分の要素数は最小となる．（証明終り）

この性質は，*ダグ順序*に行列を並べ変えることが，行列の三角化の自然な拡張になっていることを示すものである．

次に，*ダグ順序*の重要な性質として，その順に G4' の消去法を実行すると，上三角部分には新たな要素が生成されないことを示す．

消去の対象となる下三角部分の要素は，すでに述べたように後退辺に対応し，*ダグ*の一部と結合して強連結成分を構成する．これは，すでに述べた簡約集合が定める部分グラフにほかならない．後退辺 $e = (u, v)$ に対し，その終点 v によって定まる簡約集合を reduce-set(v) とする．簡約可能グラフの性質 4 により，$u \in$ reduce-set(v) である．reduce-set(v) の v 以外の頂点はすべて v によって支配されるから，1 つの*ダグ順序* S をとった時，

$$\forall u \in \text{reduce-set}(v) - \{v\}, \text{ord}(u, S) < \text{ord}(v, S),$$

が成り立つ．

さらに次の性質がいえる．

補題 5.3 *ダグ順序 S にしたがって並べられた行列に関し，後退辺 $e = (u, v)$ による reduce-set(v) に対応する行と，v より右にある列，すなわち $S^+(v)$ に対応する列で作られる部分行列で，要素が存在するのは v の行に限られる．*

証明 簡約集合が定める部分グラフとしての領域が，1つの入口（すなわち v）しか持たないという性質から明らか．（証明終り）

この補題のイメージを図示すると，図5.10のようになる．図で○で示した部分には，下辺上を除き要素が存在しない．ただし，一般のダグ順序では，reduce-set(v) は必ずしも連続した行（および列）をしめない．また，v は reduce-set(v) の中で順位がもっとも大きい（最右の列，最下の行）が，u は必ずしも順位がもっとも小さいとは限らない．

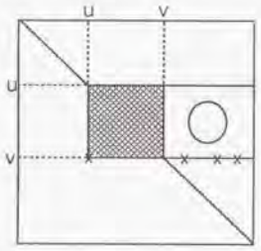


図 5.10 ダグ順序に並べられた行列上の簡約集合

以上の考察により，次の定理が成り立つ．

定理 5.2 *簡約可能フローグラフに対し，ダグに基づく順序 S に従ってアルゴリズム G_4' の消去法を行うと，上三角部分に新たな辺をまったく生成しない．すなわち，上三角部分（対角成分は除く）は変形を受けずにそのまま三角分解後の U となる．*

証明 辺 $e = (u, v)$ に対応する要素 f_{vu} (ただし $\mathrm{ord}(u, S) < \mathrm{ord}(v, S)$) を消去するとする．以下，簡単のために S に従った順位と頂点とを同一視する．すなわち，$\mathrm{ord}(v, S)$ をあらためて v とみなす（したがって，$(u < v)$）．

消去とは，$w \in \mathrm{pred}(u)$ について代入 (u, w, v) を実行することであるが，補題5.3によりそのような w は $w \leq v$ の範囲にしか存在しない（さらに，消去のアルゴリズム上，$u < w$ である）．したがって，この代入によって新たに生成される辺 (w, v) は，下三角部分（対角要素を含む）に限られる．

ここで，$w \in$ reduce-set(v) である．実際，w は u の先行点で，u から，v に支配された頂点のみを通って v に至る経路が存在するから，w が v に支配されることがいえれば，w から，v に支配された頂点のみを通って v へ至る経路が存在することがいえる．ところで，もし w が v に支配されていないとすると，辺 (w, u) によって出発点から v を通らずに u に至る経路が存在することになり，u が v に支配されているという仮定と矛盾する．

したがって，辺 $e = (u, v)$ の消去で生成された新たな要素 f_{vw} についても，その消去に対して同じことがいえる．（証明終り）

系 5.1 *簡約可能グラフにおける出発点から他の全頂点への最短路問題では，消去計算が不要である．すなわち，タグに基づく順序 S に従って行列を作ったとき，下三角部分を無視し，上三角行列をそのまま用いて，後退代入を行えばよい．*

証明 4.2.5節の議論により，最短路問題のような $f(1) = 1$ がすべての f について成立する問題では，出発点から他の頂点への経路を解析する場合に，出発点を最後のかなめとすれば前進代入が必要なくなる．しかも，定理5.2よりタグに基づく消去順序を用いれば（このとき明らかに出発点は最後におかれる），上三角部分は変化を受けず，対角部分は最短路問題では無視できるので（次数0のため，$*$ 演算が不要），系が成り立つ．（証明終り）

5.3.6 ループ順序

定理5.2が示す関係をより明確にするために，ダグ順序にさらに条件を加えた“ループ順序”を定義しよう．そのために，簡約集合をさらに細分化したループ集合を定義する．ループ集合は，後退辺 $e = (u, v)$ に対して定まる頂点集合である．これを，loop-set(e) と書く．その定義は，次式による．

$$\text{loop-set}(e) = \{w \in V | t(e) \prec w \,\&\, w \prec h(e)\}$$

ここで，$\prec$ はダグ上の半順序関係を表す．

補題 5.4 loop-set(e) は，reduce-set($t(e)$) の部分集合である．

証明 ループ集合の定義から，$w \in$ loop-set(e) に対し，w から $h(e)$ に至る経路が存在する．w が $t(e)$ に支配されることは，この経路の存在と，$h(e)$ が $t(e)$ に支配される（簡約可能グラフの性質 4）ことからいえる．

また，w から $t(e)$ に至る経路で，その上の頂点がすべて $t(e)$ に支配されるものとしては，同じく w から $h(e)$ に至る経路に e を付け加えたものをとればよい．（証明終り）

この補題から，ループ集合がたしかに簡約集合の細分化になっていることが分かる．ループ集合と簡約集合が一致しない場合として，次の 2 つがあげられる．

1. 簡約集合は後退辺の終点によって決められるが，ループ集合は後退辺によって定められている．2 つ以上の後退辺が終点を共有する時，ループ集合はそれぞれの後退辺ごとに定義されるが，簡約集合はそれらのループ集合の合併となる．

   たとえば，図 5.11 の (a) や (b) の場合，簡約集合はいずれも $\{1,2,3,4\}$ であるが，ループ集合は，(a) では $\{1,2,3\}$ と $\{1,2,4\}$，(b) では $\{1,2,3\}$ と $\{1,2,3,4\}$ となる．

2. ループ集合は，それに属する頂点から入口 ($t(e)$) へ至る経路として，ダグ上の辺のみによるものを考えている．一方，簡約集合では後退辺も含めて考えているので，図 5.12のような場合に違いが生じる．この図でループ集合は $\{1,2,3\}$ および $\{2,3,4\}$ だが，対応する簡約集合は $\{1,2,3,4\}$ と $\{2,3,4\}$ となる．

   なお，ループ集合が定める部分グラフは，ほとんどの場合領域となるが，例外的に図 5.12のループ集合 $\{1,2,3\}$ は，領域とはならない．

簡約可能グラフに対して，次の手順でえられる頂点順序をループ順序と呼ぶことにする．

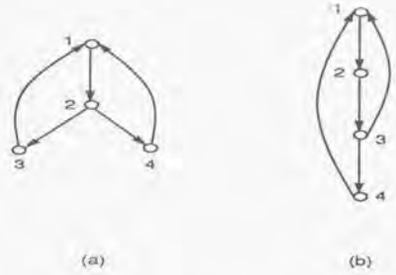


図 5.11 ループ集合と簡約集合（その 1）

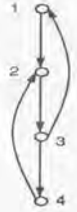


図 5.12 ループ集合と簡約集合（その 2）

ダグを作り，それで定まる後退辺の集合を B とする．

$B \neq \emptyset$ の間，以下を繰り返す．

B の要素のうち，その終点が他の B の要素の終点に支配されないものを任意に 1 つ取りだし，e とする．

loop-set(e) を求め，それらの点をすべて頂点 $t(e)$ に圧縮する．辺 e，および loop-set(e) の要素間の辺を削除し，loop-set(e) 内とその外の頂点との間の辺は，$t(e)$ との間につけ替える．頂点 $t(e)$ は，loop-set(e) を代表するという．

結果として残る無閉路グラフの頂点を位相整列し，その逆順に並べる．
頂点の並びの中に，ループ集合を代表するものがある間は，以下を繰り返す．

ループ集合を代表する頂点を，そのループ集合が構成するもとのグラフの部分グラフ(ただし，このループ集合を定める後退辺は除く．それにより，この部分グラフは，無閉路グラフになる)に関して，頂点を位相整列したものの逆順に並べた部分列に置き換える．1つの頂点が2つ以上のループ集合を代表することがあるが，その場合はたとえば圧縮した順に展開する．(展開した部分列に，さらにループ集合を代表する頂点が存在しうることに注意．)

このループ順序は，ダグ順序になっている．なぜなら，ループ集合が構成する部分グラフは1つの入口しか持たず，したがって部分グラフをその入口に圧縮しても，ダグの順序構造に影響が及ばないからである．

ループ順序によって，ループ集合が連続した部分列をなすようになるので[3]，その構造が見やすい．とくに，フローグラフがプログラムの構造に対応している場合は，プログラムのループ構造を，その入れ子関係を含めて明確に示すものとなる．このような考え方に基づいて，プログラムのループ間の関係を分類した例に，玉井[73](pp. 79–81)がある．

[例] 図5.13のグラフに対し，ダグ順序の例を図5.14に，ループ順序の例を図5.15に示す．

このループ順序の作り方は，およそ次のようである．

1. 図5.13のようにダグを定め，後退辺の集合として，$B = \{(4,2), (5,1), (11,1), (9,3)\}$ をえる．
2. 辺 $(4,2)$ を選び，そのループ集合として $\{4,2\}$ をえる．これを圧縮した頂点を $2'$ とする．

以下同様に，次のようなループ集合を得る．

[3]ただし，2つのループ集合が入口を共有したり，一方が他方(の一部)を包含する場合，必ずしも単純に連続した部分列とはならない．

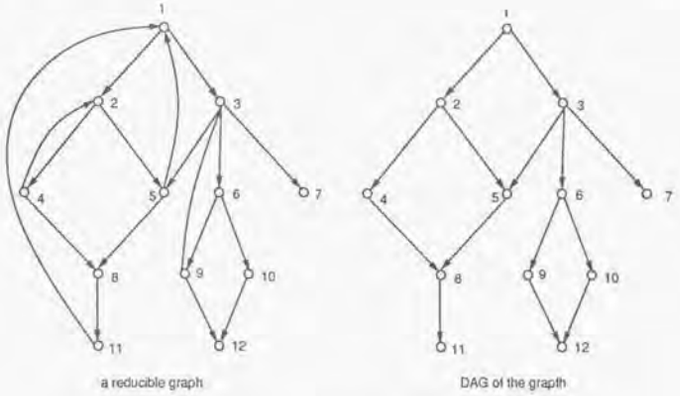


図 5.13 いくつかのループ集合を持つ簡約可能グラフとそのダグ

後退辺	ループ集合
$(9,3)$	$3' = \{9,6,3\}$
$(5,1)$	$1' = \{5,3',2',1\}$
$(11,1)$	$1'' = \{11,8,1'\}$

3. 残った無閉路グラフを位相整列の逆順に並べ $(12,10,7,1'')$ をえる．さらに，$1''$, $1'$, $3'$, $2'$ を順に展開して，最終的に次をえる．

$12,10,7,11,8,5,9,6,3,4,2,1$

同じループ順序を，次に示すような方法で作ることもできる．これは上の方法が内側の小さなループから見つけていくという，いわばボットムアップ的であったのに対し，外側のループから見つけていくという意味で，トップダウンの方法といえる．

[トップダウンによるループ集合への分解]

1. グラフの強連結成分分解を行い，その成分を到達可能性による半順序に関して位相整列した順序の逆順に並べる．

	11	8	4	5	2	12	9	10	6	7	3	1
11	○	×										
8		○	×	×								
4			○		×							
5				○	×						×	
2			×		○							×
12						○	×	×				
9							○		×			
10								○	×			
6									○		×	
7										○	×	
3							×				○	×
1	×			×								○

図 5.14 ダグ順序の一例

2. できた各強連結成分には，簡約可能グラフの性質 1 から，ただ 1 つの入口しか存在しない．その入口を出発点として，その出発点に入る辺をまず削除する．ただし，もとのフローグラフの出発点を含む成分については，その出発点を強連結成分の出発点とする．各成分に対し再帰的に 1 を実行し，得られた部分順序列を，もとの順序列の対応する成分のところに展開する．これを，すべての成分が 1 頂点になるまで続ける．

この過程で現れる強連結成分が，ループ集合の定める部分グラフに対応する．ただし，強連結成分の出発点に入る辺が複数ある場合に，それらを一度に削除すると，図 5.11 によって述べたような意味で，簡約集合には対応しても，ループ集合に対応する強連結成分が現れないことが起こりうる．したがって，上の手順で 1 つの辺ずつ削除しないと，ボトムアップの方法に正確には対応しない．

この方法は，ループ順序を求めることが強連結成分によるブロック三角化の拡張になっていることを示すものである．

	12	10	7	11	8	5	9	6	3	4	2	1
12	○	×					×					
10		○						×				
7			○						×			
11				○	×							
8					○	×				×		
5						○			×		×	
9							○	×				
6								○	×			
3							×		○			×
4										○	×	
2										×	○	×
1				×		×						○

図 5.15 ループ順序の一例

5.3.7 ループ順序による消去法の手間の評価

定理5.1, 5.2から，簡約可能グラフ上の問題に対しては，最短路問題に限らず，一般にダグ順序あるいはループ順序による消去が有効であることが分かる．これを，さらに評価してみよう．

ループ順序による消去法の手間を評価するために，まずループ集合のうち，それを作る後退辺の終点を共有するものは，1つにまとめたような集合クラス Q を考える．すなわち，

$$Q = \{q \in 2^V | \exists v \in V.\, q = \bigcup_{t(e)=v} \text{loop-set}(e)\}.$$

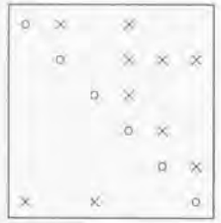
形式的には Q に空集合 $\emptyset$ が含まれるが，以下では簡単のため Q から $\emptyset$ を除いたものをあらためて Q とする．Q の要素はほぼ簡約集合に対応するが，図5.12のように2つのループが入れ違いになっている場合にのみ異なる．本節では，この Q の各要素をループ集合と呼ぶことにする．

Q の要素 q が定める部分グラフで，G_U に属する辺の数を $m_{U,q}$ とする．簡約可能グラフにダグ順序やループ順序による消去法を適用したとき，G_U の辺の数が増加しないことに注意しよう．簡約可能グラフに対してループ順序により消去法を適用した場合，代入操作による関数合成の回数の上限は，

$$\sum_{q \in Q} m_{U,q} \tag{5.1}$$

で与えられる．このことは，Q の要素が行列上で作る正方形に対し，掃き出しの対象となるのは下三角部分の底辺の行のみであり，その掃き出しに対し，上三角部分にある要素の数だけの関数合成が必要となることから分かる（図5.16参照）．なお，還元操作に伴う関数合成の回数はたかだか G_U の辺の数 m_U であり，後退代入に伴う関数合成の回数はちょうど m_U となる．この評価は，ループ順序だけでなく，ダグ順序にも成立する．

(5.1) 式の上限は，$m_{U,q} \leq |q|(|q|-1)/2$ であり，$|Q| \leq n$ であるから，全体として $O(n^3)$ となる．これは一般のガウス消去法の手間のオーダーであるから，かなり過大評価である．実際，最悪のケースは行列として完全に密な場合，グラフでは完全グラフに対応する場合であるが，完全グラフは簡約可能ではないから，効率に関しもう少しきめの細かい議論ができるはずである．



このループ集合の2つの後退辺を消去するには，$m_{U,q} = 8$ 回の関数合成が必要．

図5.16 ループ集合の下三角部分の消去の手間

まず極端なケースとして，上三角部分が完全に密なグラフを考えてみよう．この場合，簡約可能性を持つためには，図5.17のように，後退辺はすべて G_U の根（自分自身には入る辺がなく，他のすべての頂点に到達可能な頂点）を終点にするものに限られる．この場合明らかに，関数合成の回数は $O(n^2)$ である．

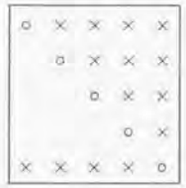


図5.17 上三角部分が密な簡約可能グラフ

上の場合は $|Q| = 1$ となるが，逆に $|Q| = n$ となる場合は，上三角部分のグラフが連結ならば木，一般には森になる，すなわち辺の数がたかだか $n-1$ であることが，簡単に示せる．たとえば，図5.18の場合である．この場合も手間は $O(n^2)$ である．

式 (5.1) が $O(n^2)$ より高いオーダーになる可能性は，ループ集合に共通部分

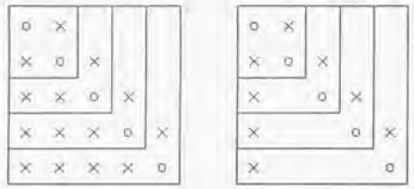


図 5.18 $|Q| = n$ の場合の簡約可能グラフ

がありうることから生ずる．もし，Q の各要素が互いに素なら，手間はもちろん $O(m)$ になるが，ループ集合の重なりの数が n によらない定数 k で押えられる場合も，$O(m)$ となる．この仮定は，プログラムのフローグラフに対してはそれほど無理な仮定ではない．実際，プログラムのループの入れ子の深さに上限が決められることは実験的に確かめられている．Allen & Cock [5] のアルゴリズムが $O(n^2)$ といわれるのは，この性質を用いている [43]．あるいは，これを仮定しない場合には辺の数 $m = O(n)$ を仮定している [58]．

図 5.17や 5.18をみると，ループ集合に重なりがあっても，一般に $O(n^2)$ がいえそうに見えるが，図 5.19のような例がある．

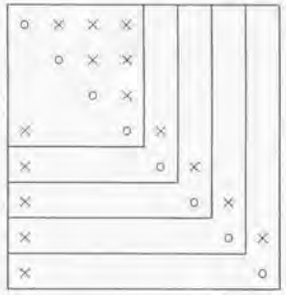
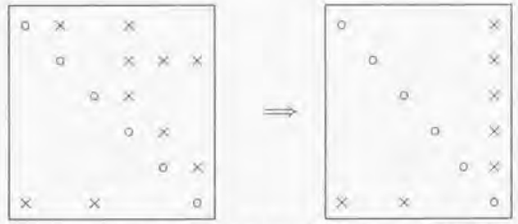


図 5.19 手間が $O(n^3)$ となる簡約可能グラフ

すなわち，一番内部のループ集合が $n/2$ の大きさを持ちかつ上三角部分が密であって，しかもループ集合が $n/2$ 重に入れ子になっていると，手間は $O(n^3)$ とならざるをえない．

しかし，ここでループ順序による消去法のアルゴリズムに，簡約順序で使われる上三角部分も適当なタイミングで掃き出すという手法を加味すると，$O(n^2)$ オーダーが実現できる．実際，一番内側の頂点 v_q を入口とするループ集合 q に対し，図 5.20に示すように，$m_q - m_{q,0}$ の手間で，v_q から出る辺以外の上三角部分を掃き出すことができる．ここで，$m_{q,0}$ は v_q を始点としループ集合 q に含まれる頂点を終点とする辺の数である．その後，後退辺を掃き出す手間は，そのループ集合に含まれる後退辺の数に比例する．ただ，この操作で，v_q から出る辺の数が増加しうる．しかし，その上限は $|q| - 1$ で押えられる．



$m_{U,q} = 8, m_{q,0} = 2$ だから左から右への変形は 6 回の関数合成でできる．その際，新たに 3 つの辺が生成される．2 つの後退辺の消去は，それぞれ 1 回の関数合成でできる．

図 5.20 $O(n^2)$ の手間による上三角部分の消去

このような消去を内側のループ集合から順に行なえばよい．ループ順序に従えば，自然に内側のループから消去が実行できる．全体に要する計算の手間は，関数合成で数えて，

$$m - \sum_{q \in Q} m_{q,0} + \sum_{q \in Q} |q| \tag{5.2}$$

すなわち，全体の辺の数 m に対し，増加分はループ集合の入口を始点とする辺で新たに生成されたものである．上式の上限は明らかに $O(n^2)$ である．なお，この場合上三角部分も消去されるので後退代入は実質的に不要となる．

従来 $O(n^2)$ とされてきたアルゴリズムは，$m = O(n)$ を仮定するものであったが，式 (5.2) の評価はそれを仮定していない．簡約順序のアルゴリズムが $O(m \log n)$ であるのと比較して，$m = O(n)$ の場合は劣るが，$m = O(n^2)$ の場合は優るものとなる．

5.3.8 例題による効率の評価

例題により簡約順序とループ順序を比較してみよう．ここでは，ループ順序による消去法は前節で述べた $O(n^2)$ に変更したものではなく，もとのアルゴリズム G4' の形式のものとする．

まず，簡単な例として，Tarjan[56] で使われているものを取り上げる．グラフと簡約順序およびループ順序による行列表現は，それぞれ図 5.21, 5.22の通り．

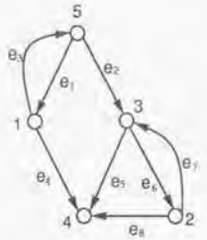


図 5.21 簡単な簡約可能グラフによる比較

このダグ順序に G4" アルゴリズムを適用すると，3回の関数合成 (内2回は f_{32}(辺 e_7) および f_{51}(辺 e_3) の消去に伴うもの，1回は，f_{33} の還元操作に伴う $f_{33}^* \cdot f_{35}$)，2回の不動点関数生成が必要となる．さらに，後退代入で，6回の関数合成[4]と2回の交わり演算が必要となる．

[4]通常，後退代入では関数合成ではなく関数適用が行なわれるが，ここでは Tarjan の例題の趣旨に合わせて，それを関数合成として考える．具体的な値を想定せず，関数適用を式として計算すると考えれば同じことになる．

簡約順序

	1	2	3	4	5
1	○				×
2		○	×		
3		×	○		×
4	×	×	×	○	
5	×				○

ループ順序

	4	2	3	4	5
4	○	×	×		×
2		○	×		
3		×	○		×
1				○	×
5				×	○

図 5.22 簡約順序とループ順序

一方簡約順序の場合を Tarjan の方法にしたがって実行すると，消去に相当する前半部分で，関数合成7回と不動点関数生成2回，および2回の交わり演算が行なわれる．その後，後退代入に相当する部分で，4回の関数合成が必要となる．

関数合成で比較すると，ループ順序で9回なのに対し，簡約順序に基づく Tarjan の方法では11回となる．もちろん，このような小さな例で効率の優劣を評価するのは意味がないが，少なくとも簡約順序に基づく方法が常に最適とはいえないことは，注意すべきであろう．

もう少し大きな例として，図5.13を考えよう．

ループ順序による消去の結果生じる要素を行列で示すと，図5.23のようになる．ここで，$\not{\checkmark}$ は，代入操作によって，もともと辺が存在する2頂点の間に新たな辺が生成された場合を示している．実際は合併操作によって1つの辺にまとめられる．

必要な関数合成は，消去で11回，後退代入で14回の計25回，不動点関数生成は3回である．

一方，簡約順序として

$$4, 2, 9, 10, 12, 6, 7, 3, 5, 11, 8, 1$$

をとり，Tarjan の方法を実行すると，関数合成は28回となる (不動点関数生成は同じ3回)．この回数のカウントはやや複雑であるが，まず消去に相当する部分で，12回の関数合成が起こる．その結果，図5.24のような木ができる．この木から後退代入に相当する操作 (その際にも木の高さを縮める変形が起こ

	12	10	7	11	8	5	9	6	3	4	2	1
12	○	×					×					
10		○						×				
7			○						×			
11				○	×							
8					○	×				×		
5						○			×		×	
9							○	×				
6								○	×			
3							×	√	⊗			×
4										○	×	
2										×	⊗	×
1				×	√	√			√	√	√	⊗

図 5.23 消去結果

る) を実行するのに，16回の関数合成が必要となる．

この例でも，ループ順序の方が少ない手間ですんでいる．もちろん，この2つの方法の効率を比較するには，より大きなグラフによる統計的な検証が必要であろう．しかし，ループ順序で単純なガウス型の消去法を適用することは，たとえば森の操作による木の高さの圧縮とか，2-3木の保持といった複雑なデータ処理を伴わないだけに，魅力がある．上の簡単な例は，このループ順序による方法の妥当性を示唆するように思われる．

5.4 簡約不可能なグラフへの拡張

ダグ順序やループ順序の定義は，簡約不可能なグラフに対してもほぼそのまま適用できる．すなわち，無閉路で極大な部分グラフというダグの定義は一般的に成り立つから，ダグを1つ定めれば，それに対してダグ順序が定まる．ただし，ダグの取り方には任意性が残る．また，ループ順序については，ボトムアップの手順は，同様にダグを1つ定めて，以下はまったく同じ方法をとれば

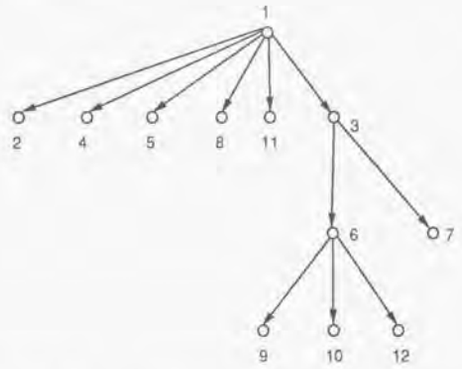


図 5.24 簡約順序による消去で生成されるヘッダ木

よい．また，トップダウンの手順でいえば，強連結な部分グラフに入口が2つ以上ある場合に，そのどれを出発点として選ぶかという任意性があるが，適当な1つを選ぶことにより，やはりループ構造を反映した順序が得られる．こうして得られる順序をやはりループ順序と呼ぶことにする．

ダグの選び方や強連結な部分グラフの出発点の定め方によって，作成される頂点順序が異なってくるだけでなく，下三角部分にできる要素の数も一般には異なることがありうる．すなわち，定理5.1および5.2はもはや成立しないが，その系として次の性質が成り立つ．

系 5.2 必ずしも簡約可能でないフローグラフをダグ順序に並べた行列表現では，下三角部分にある任意の辺を上三角部分で構成されるダグにつけ加えれば閉路ができる，という意味で，下三角部分の要素数は極小である．

系 5.3 必ずしも簡約可能でないフローグラフをループ順序でガウス式に消去した場合 (アルゴリズム *G4'*)，行列表現の上三角部分に新たに生成される辺は，2つ以上の入口があるループ集合に対して，対応する強連結部分グラフの出発点として選んだ頂点の行を，他の入口の行で掃き出す場合に，出発点の行に生成されうるものに限られる．

この系5.3の表現は分かり難いかもしれないが，後で示す例によって，その意味は明らかとなろう．

簡約可能でない場合のダグ順序またはループ順序を求めるのに，前述のような不定性があるが，その選択基準としてたとえば次のようなものが考えられる．ここでは，トップダウンにループ順序を作る場合の，強連結部分グラフの出発点の選び方という点について考える．

1. 消去前の下三角部分の要素数をなるべく減らすことを目的とする場合．
   ループ集合に対応する強連結部分グラフに複数の入口があるとき，1つの入口から，その入口に入る後退辺(ループ集合の内部の頂点からその入口に入る辺)の始点に至る，閉路を含まない経路の数が最も多いものを出発点に選ぶ．入口に入る後退辺が複数ある時には，各辺の始点への経路の数の総計をとる．

2. 消去に際して，上三角部分の辺の生成をなるべく減らすことを目的とする場合．
   ループ集合に対応する強連結部分グラフに複数の入口がある時，入ってくる辺の数が最大の入口を，出発点として選ぶ．

ただし，どちらの戦略もヒューリスティックなもので，最適性は保証されない．その有効性については，さらに検証が必要であろう．

簡約不可能なグラフにループ順序による消去を実行した例を次に示す．この例も，Tarjan[56]からとった．Tarjanも簡約が必ずしも可能でないグラフに対して，その方法を拡張しているが，その手法はきわめて複雑である．ループ順序による方法の利点は，その順序を定める際に簡約可能性が判定でき，しかも簡約不可能な場合も，自然な拡張で順序が決められること，および順序が定まった後の消去法の適用に関しては，簡約可能か否かをまったく気にしなくてよい点にある．

[例] 図5.25にグラフを示す．

ループ順序とそれにしたがった消去結果を示したのが，図5.26である．ここで，$\sqrt{/}$などは，同じ頂点対の間に2度（以上）辺の生成が起こることを示している．とくに，$\sqrt{\sqrt{/}}$は，1つの頂点にセルフループが3度生成されたことを示す．

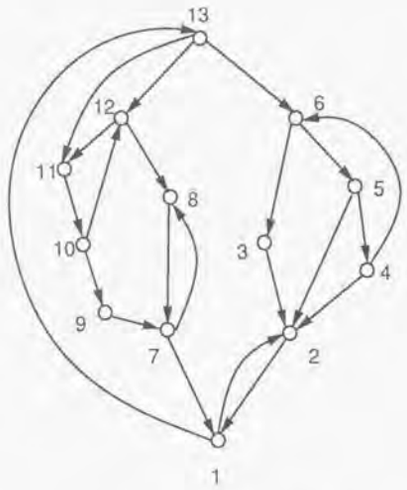


図5.25 簡約不可能グラフ

3つのループ集合，(1,2),(7,8),(10,11,12)は，それぞれ2つの入口を持つ．そのため，上三角部分に(7,2),(9,8),(13,12)という辺が生成される（行列の表記で$\sqrt{}$で表している．ただし，(13,12)には，最初から辺が存在するので，実際には辺の生成ではなく合併が起こる）．

この消去法では，消去過程で上述の3つの辺の生成に伴う関数合成3回，下三角部分の辺の生成に伴うもの25回，還元操作に伴うもの8回の，総計36回の関数合成が起こる．このうちとくに目だつのは，行13に生成される12個の辺を作る際の関数合成である．不動点関数生成は5回（後退辺の終点となる頂点数）である．

後退代入では，19回（上三角部分の要素数）の関数合成が必要となる．

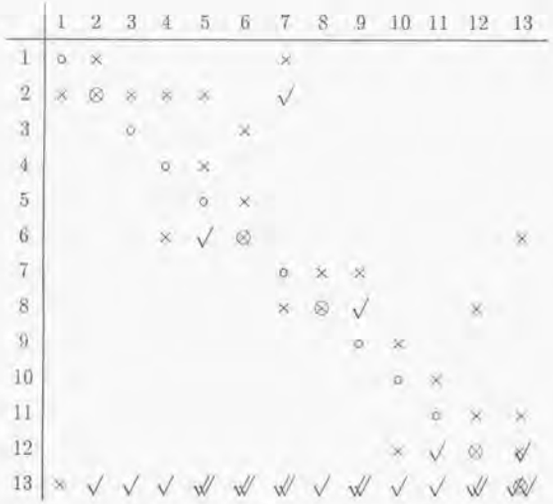


図 5.26 ループ順序による消去結果

5.5 出発点のもつ意味

ここで，フローグラフにおける出発点の意味について考えてみよう．もちろん，最短路問題やデータフロー解析問題では，もともとの問題の意味から出発点という概念が自然に導き出される．しかし，抽象化したグラフ上で考えた場合，次のような疑問が浮かび上がる．

1. 第3章の逐次法では出発点の存在を前提にし，第4章の消去法では出発点の存在を仮定しなかった．主に簡約可能グラフを扱うこの章では，ふたたび出発点を前提にしているが，これらの間にはどういう関係があるか？

2. 第5章の範囲でも，強連結成分分解によるブロック三角化では出発点の存在は仮定せず，簡約可能なグラフを扱うときに，出発点を想定した．グラフの簡約可能性は，どうして出発点に依存した定義になっているのか？

3.4節で述べたように，方程式(2.1)の不動点解は，出発点から任意の頂点へ至るあらゆる経路に沿って，経路上の辺のラベルとなっている関数を適用した結果の交わりと，（一定の条件のもとで）解釈できる．したがって，出発点という概念は，問題の解釈上は本質的なものといえ，出発点を前提としないで考える場合は，単にあらゆる頂点を出発点とするように一般化して考えていると見ることができる（最短路でいえば全頂点対間の最短路問題）．また，1.1節で詳しく論じたように，出発点を前提とする定式化と出発点を特定しない定式化とは，適当な操作によって互いに変換することができる．

ただ，出発点を前提とするか否かは，アルゴリズムにも影響する．逐次法の場合は，アルゴリズム上の要請から出発点を仮定する．消去法の場合は，方程式(1.2)を取り扱っていることからも分かるように，任意の定数ベクトル b について解くことを前提にしており，出発点は任意でよい．

しかし，いったん消去順序を考慮することにすれば，消去順序で最後におかれる頂点に出発点という意味づけを与えるのが自然になる．その理由の1つは，4.2.5節に述べたように，定数ベクトル b が最後の頂点に対応するところにのみ要素を持つとしたら，ガウス消去法に対応する計算に際して，前進代入に相当する操作がいらないからである．また，行列の三角化あるいはブロック三角化ができる場合，それにしたがって並べた頂点順序の最後の点（ブロック三角化で，最後のブロックが2つ以上の頂点を持つ場合は，その中の任意の点）は，その頂点から他のすべての頂点に到達可能という意味で，やはり出発点という意味づけが自然となる．

簡約可能グラフの定義における T2 変形では，吸収される頂点の対象から出発点を除いている．したがって，図5.2に示したように，同じグラフでも出発点のとり方によって簡約可能になったりならなかったりする．もし出発点が指定されていなければ，簡約可能となるような出発点を（もしあれば）選んで，それに基づいて簡約順序なりダグ順序なりを定める，という方法は考えられる．この場合は，グラフの強連結分解と簡約可能グラフのループ構造の分解とが，類似の操作となる．トップダウンによるループ順序の決定は，両者の関係を示すものといえる．

第 6 章

まとめと関連する課題

これまで述べたことを総括し，とくに本論文における問題の見方と方法の特徴をまとめる．また，今後の研究課題として，並列計算モデルと AND/OR 型の問題群への拡張という 2 つのテーマを取り上げる．

6.1 まとめ

最短路問題やデータフロー解析に代表されるような，グラフ上のデータの流れによって計算が進むという構造をもった問題が，グラフ上の不動点問題として統一的に定式化できることを示し，それに対する実用的な解法を，大きく逐次法と消去法に分けて示した．その特徴と成果をまとめると，次のようになる．

- 問題の定式化に際しては，なるべく一般性が高いものを目指すとともに，その基本的な構造が明確になることを主眼とした．したがってとくに，問題の構造が持つある種の双対性と，グラフの位相的構造からくる双対性との関係に注意した．

- 多様な問題を統一的な枠組によってとり扱えるようにすることは，本研究の大きな目的であるが，同時に抽象化のための抽象化となるような危険は避けるように努力した．したがって，一般的な枠組によって議論した問題の性質や解法に対しても，具体的な問題における意味づけやそれへの適用について，できるかぎり言及した．

- アルゴリズムに関しては，計算の複雑性を尺度とする評価だけでなく，実用性を重視した．実用性とは，アルゴリズムの実現のしやすさ，疎な性質を持つ問題に対しての，平均的な効率のよさ，などの性質である．実際的な問題に対処したり，そのためのソフトウェアシステムを開発した経験から，これらの重要性を感じてきたからである．

  したがって，たとえば逐次型の解法では，II アルゴリズムとして，ある意味でもっとも素直な，しかし分野によっては正しく認識されていない方法を詳しく考察したし，消去型の解法では，基本的なガウス消去型のアルゴリズムを，消去順序の工夫によっていかに効率を上げるかという観点から主に論じた．

- 解法の実用性と関連して，問題の疎な性質を利用するという観点から，グラフ的な処理を行列的な処理より一貫して重視した．同時に，行列上で工夫されてきた消去順序のような技法と，グラフ上の操作との対応について，明確な視点を与えた．

- 最短路問題やデータフロー解析など，様々な分野で研究されきた問題を，統一的に取り扱うという本論文の性格上，ここで述べた多くの概念や性質は，すでに個別の分野で何らかの形で知られていたものを，再整理，再構築したものであるという面がある．その中で，新しい知見と思われる主なものとしては，次が挙げられよう．

  - 逐次型の解法を一般的な設定のもとに与え，その正当性を証明したこと．
  - データフロー解析の分野でのみ論じられてきた簡約可能グラフの性質を，消去型解法における消去順序との関係，およびグラフの強連結成分分解との関係によって，明らかにしたこと．
  - 消去型の消去順序に関して，実際的な有効性の認められる提案をしたこと．その解法は，簡約可能グラフ上の最短路問題にはきわめて効率がいいが，まったく同じ形で簡約不可能な一般グラフ上の最短路以外の一般的な問題に対しても有効であるという特徴を持つ．

6.2 並列計算モデル

今後の研究課題として，まず並列計算モデルを取り上げる．

グラフの各頂点をプロセスと考えれば，ここで述べた計算システムを次のように解釈することができる（図6.1参照）．

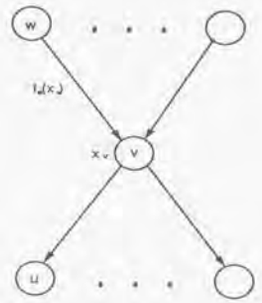


図 6.1 並列プロセスとしての解釈

1. 各プロセス v は，他のプロセス w から，チャネル $e = (w, v)$ を介してメッセージ $f_e(x_w)$ を受け取る．
2. プロセスは自己の状態 x_v と $f_e(x_w)$ とを評価し，適当な条件（$x_v \leq f_e(x_w)$ でない）が成立すれば状態を（$f_e(x_w) \wedge x_v$ に）変更し，かつ自分が送信すべき相手のプロセス u ごとに，状態変化に応じたメッセージをチャネル (v, u) を介して送る．

このように解釈すると，われわれの定式化は，簡素な並列計算モデルを規定したものとみなせる．アルゴリズム II は，それを単一プロセッサで処理する場合のスケジューリング算法に相当する．

第2章で述べたように，われわれの定式化は，正規表現と有限オートマトンのモデルを包含する．したがって，基本的な並列動作モデルとしての有限オートマトンとは，自然な対応関係がある．ペトリネットについては，後で簡単に

触れるように，束の∧と∨の両演算を結合するような拡張されたモデルを考えると対応がつく．

より具体的な既存のモデルで関連が深いものに，並列論理型プログラミングのモデルがある [45, 57, 62]．たとえば，[62] には最短路問題のプログラムがあるが，その処理方法はちょうどこの解釈に相当する．ただし，計算の終了を検出するための技法が導入されている．

並列オブジェクト指向のモデルも，基本的にメッセージの交換による並列動作をモデル化しているので [77]，このグラフ上の動作モデルと適合する．今後，さらに研究する意味があろう．

これまで束の構造を前提としてきたが，第2章でも論じたように，束の半順序構造は計算を収束させるために必要とする性質であった．計算が停止した時に結果が得られるという逐次型の計算システムでは，このように停止を保証するための仕組みが必要であるが，並列計算システムでは，そのプロセスに意味があり，むしろ停止しないことが普通であるともいえる（もちろん，並列に計算して，全体が停止した時に結果が出るというモデルもある．この辺の議論については，[45] 参照）．したがって，並列システムを扱うには，本論文における定式化も，そのもっとも基本的な構造に還元し，可換で結合的な∧と結合的な・という演算のみを土台として，理論を構築していく方向が展望される．

6.3 AND/OR 型への拡張

最短路問題に対するダイクストラ法を，Knuth が一般化している [33]．その一般化は，グラフの頂点に入る辺の結合の種類を，AND 型と OR 型の2種類に拡張したものと解釈できる．Knuth は，最短路問題のような全順序性を持つ数値を値域とするものに対象を限定しているが，この着想をさらに拡張して，ここで述べた枠組みの中に取り込むことが考えられる．

われわれの定式化では，情報の代数的な構造を表すのに束を用いているが，束の2つの演算のうち実質的には一方しか使わなかった．Knuth の方法は，次のように∧と∨の両演算を対等に用いる定式化を示唆する．

$$x_v = \bigwedge_{e_i \in \mathrm{Andin}(v)} f_{e_i}\Big(\bigvee_{e_j \in \mathrm{Orin}(h(e_i))} g_{e_j}(x_{h(e_j)}) \Big). \qquad (6.1)$$

ここで Andin は AND 結合の頂点へ入る辺の集合，Orin は OR 結合の頂点へ入る辺の集合を表す．AND 結合と OR 結合は交互に接続されているものとする．

式 (6.1) は，AND 型（∧を AND と解釈するとすれば）の頂点間の関係を示したものであるが，これを AND 型と OR 型の頂点に分けてそれぞれ別に示せば，次のようになる．

$$x_v = \bigwedge_{e \in \mathrm{Andin}(v)} f_e(x_{h(e)}) \quad (v \text{ は AND 型の頂点}). \qquad (6.2)$$

$$x_w = \bigvee_{e \in \mathrm{Orin}(w)} g_e(x_{h(e)}) \quad (w \text{ は OR 型の頂点}). \qquad (6.3)$$

∧を min 演算に，∨を max 演算に対応させれば，上式はゲームの理論などで現れる min-max 問題に対応する．そのほか，AI における AND/OR 木の探索，項書き換えシステム，Prolog のような論理型プログラミング，形式言語システム，などとも対応がとれるので，それらの問題分野のある種の解析にここで述べた方法が活用できる可能性がある．実際，伊理による束を用いた定式化はこの AND/OR 型を含んでおり，応用例として2人ゲームなどを扱っている [24, 25]．論理型言語を完備束値を持つように拡張した森下 [76] の研究なども関連があるが，実質的には束演算の内のやはり一方のみを扱っており，本論文の第1章における定式化のある意味で双対になっている．

並列システムの解析モデルとしてよく用いられるペトリネットは，プレースおよびトランジションと呼ばれる，2種類の頂点を持つグラフの上で定式化される [39]．Petri によるもともとの定式化は，トランジションを AND 型の頂点に，プレースを OR 型の頂点に対応させ，束として 0-1 のブール束をとった場合に対応する．ただし，トランジションが発火すると，その入力となるプレースのトークンは取り去られる．この対応をつけるには，トランジションから入力プレースへの逆向きの辺をつけて，そのラベルは0を1に，1を0に変換する否定関数とし，さらにプレースへの結合を AND 型にするために頂点を追加するというような，やや複雑な工夫がいる．

いずれにせよ，AND/OR 型に拡張することによりさらに広い問題の範囲を取り扱いうるが，対象とする束や関数空間の性質，解法アルゴリズムなど，今後研究すべき課題は多い．

参考文献

[1] Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, pp. 195–206 (1974).

[2] Aho, A.V. and Ullman, J.D.: *Principles of Compiler Design*, Addison-Wesley (1977).

[3] Allen, F.E.: A Basis for Program Optimization, *Inf. Process. 71*, North-Holland, pp. 385–390 (1972).

[4] Allen, F.E. and Cock, J.: Graph-Theoretic Constructs for Program Control Flow Analysis, IBM Research Report RC3923, Thomas J. Watson Research Center, Yorktown Heights, N.Y. (1972).

[5] Allen, F.E. and Cock, J.: A Program Data Flow Analysis Procedure, *Comm. ACM*, Vol. 19, No. 3, pp. 137–147 (1976).

[6] Backhouse, R.C. and Carré, B.A.: Regular Algebra Applied to Path-finding Problems, *J. Inst. Math. Applic.*, Vol. 15, pp. 161–186 (1975).

[7] Bellman, R.E.: On a Routing Problem, *Quarterly Appl. Math.*, Vol. 16, pp. 87–90 (1958).

[8] Carré, B.A.: An Algebra for Network Routing Problems, *J. Inst. Math. Applic.*, Vol. 7, pp. 273–294 (1971).

[9] Cousot, P. and Cousot, R.: Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, *4th POPL*, pp. 238–252 (1977).

[10] Denardo, E.V. and Fox, B.L.: Shortest-Route Methods: 1. Reaching, Pruning, and Buckets, *Oper. Res.*, Vol. 27, No. 1, pp. 161–186 (1979).

[11] Dijkstra, E.W.: A Note on Two Problems in Connextion with Graphs, *Numer. Math.*, Vol. 1, pp. 269–271 (1959).

[12] Dreyfus, S.E.: An Appraisal of Some Shortest-Path Algorithms, *Oper. Res.*, Vol. 17, No. 3, pp. 395–412 (1969).

[13] Dulmage, A.L. and Mendelsohn, N.S.: On the Inversion of Sparse Matrices, *Math. Comput.*, Vol. 16, pp. 494–496 (1962).

[14] Floyd, R.: Algorithm 97: Shortest Path, *Comm. ACM*, Vol. 5, No. 6, p. 345 (1962).

[15] Fong, A., Kam, J. and Ullman, J.: Application of Lattice Algebra to Loop Optimization, *2nd POPL*, pp. 1–9 (1975).

[16] Ford, L.R. and Fulkerson, D.R.: *Flows in Networks*, Princeton University Press, Princeton (1962).

[17] Graham, S. and Wegman, M.: Fast and Usually Linear Algorithm for Global Data Flow Analysis, *J. ACM*, Vol. 23, No. 1, pp. 172–202 (1976).

[18] Hecht, M.S.: *Flow Analysis of Computer Programs*, North-Holland (1977).

[19] Hecht, M.S. and Ullman, J.D.: Flow Graph Reducibility, *SIAM J. Comput.*, Vol. 1, No. 2, pp. 188–202 (1972).

[20] Hecht, M.S. and Ullman, J.D.: Characterization of Reducible Flow Graphs, *J. ACM*, Vol. 21, No. 3, pp. 367–375 (1974).

[21] Hecht, M.S. and Ullman, J.D.: A Simple Algorithm for Global Data Flow Analysis Problems, *SIAM J. Comput.*, Vol. 4, No. 4, pp. 519–532 (1977).

[22] Hellerman, E. and Rarick, D.C.: The Partitioned Preassigned Pivot Procedure (P4), in Rose, D.J. and Willoughby, R.A. eds.: *Sparse Matrices and their Applications*, Plenum-Press, New York, pp. 67–76 (1972).

[23] Imai, H. and Iri, M.: Practical Efficiencies of Exsiting Shortest-Path Algorithms and a New Bucket Algorithm, *J. Oper. Res. Soc. Japan*, Vol. 27, No. 1, pp. 43–57 (1984).

[24] Iri, M.: Fundamentals of the Algebraic and Topological Treatment of General Information Networks, *RAAG Memoirs of the Unifying Study of Basic Problems in Engineering and Physical Sciences by means of Geometry*, Vol. III, Division G, pp. 418–450 (1962).

[25] Iri, M.: Several Applications of the Basic Theory of General Information Networks – Connextion Properties of Graphs and Finite Deterministic Two-Person Games, *RAAG Memoirs of the Unifying Study of Basic Problems in Engineering and Physical Sciences by means of Geometry*, Vol. III, Division G, pp. 451–471 (1962).

[26] Iri, M.: *Network Flow, Transportation and Scheduling: Theory and Algorithms*, Academic Press (1969).

[27] Iri, M.: Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors — Complexity and Practicality, *Japan J. Appl. Math.*, Vol. 1, No. 2, pp. 223–252 (1984).

[28] Johnson, D.B.: Efficient Algorithm for Shortest Paths in Sparse Networks, *J. ACM*, Vol. 24, No. 1, pp. 1–13 (1977).

[29] Kam, J.B. and Ullman, J.D.: Global Data Flow Analysis and Iterative Algorithms, *J. ACM*, Vol. 23, No. 1, pp. 158–171 (1976).

[30] Kam, J.B. and Ullman, J.D.: Monotone Data Flow Analysis Frameworks, *Acta Inf.*, Vol. 7, pp. 305–317 (1977).

[31] Kennedy, M.W.: Node Listing Applied to Data Flow Analysis, *2nd POPL*, pp. 10–21 (1975).

[32] Kildall, G.A.: A Unified Approach to Global Program Optimization, *1st POPL*, pp. 194–206 (1973).

[33] Knuth, D.: A Generalization of Dijkstra's Algorithm, *Inf. Proc. Letters*, Vol. 6, No. 1, pp. 1–7 (1977).

[34] Lengauer, D. and Theune, D.: Path Problems with General Cost Criteria in EMC-Driven Wiring, *Extended Abstracts of the First Workshop on Combinatorial Optimization in Science and Technology*, DIMACS Technical Report 91-18, Rutgers University, pp. 213–221 (1991).

[35] Milner, A. J. R. G.: *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, 92, Springer-Verlag (1980).

[36] Nelson, G. and Oppen, D. C.: Fast Decision Algorithms Based on Union and Find, *Proc. Annual IEEE Symp. Foundation of Comp. Sci.*, pp. 114–119 (1977).

[37] Nilsson, N.J.: *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill (1975).

[38] Pape, U.: Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem, *Math. Program.*, Vol. 7, pp. 212–222 (1974).

[39] Peterson, J. L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall (1981).
邦訳，市川惇信，小林重信 訳，ペトリネット入門 — 情報システムのモデル化，共立出版 (1984).

[40] Rose, D.J. and Tarjan, R.E.: Algorithmic Aspects of Vertex Elimination on Directed Graphs, *SIAM J. Appl. Math.*, Vol. 34, pp. 176–197 (1978).

[41] Rosen, B.K.: Monoids for Rapid Data Flow Analysis, *SIAM J. Comput.*, Vol. 9, pp. 159–196 (1980),

[42] Rote, R.: Path Problems in Graphs, in Timhofer, G. et al. eds.: *Computational Graph Theory*, Springer-Verlag, pp. 155–189 (1990).

[43] Ryder, B.G. and Paull, M.C.: Elimination Algorithms for Data Flow Analysis, *ACM Compul. Surv.*, Vol. 18, No. 3, pp. 277–316 (1986).

[44] Salomaa, A.: Two Complete Axiom Systems for the Algebra of Regular Events, *J. ACM*, Vol. 13, No. 1, pp. 158–169 (1966).

[45] Shapiro, E.: The Family of Concurrent Logic Programming Languages, *ACM Comput. Surv.*, Vol. 21, No. 3, pp. 413–510 (1989).

[46] Shier, D.R.: *Network Reliability and Algebraic Structures*, Oxford University Press (1991).

[47] Tamai, T. and Fukunaga, K.: A Verification System for File Processing Programs, *Proc. 3rd RIMS Symposium on Mathematical Methods in Software Science and Engineering*, Kyoto (1981).

[48] Tamai, T.: A Simplifier for Program Verification with Built-in Knowledge on Equality and Partial Ordering and its Use for Finding Loop Invariants, *J. Inf. Proc.*, Vol. 6, No. 4, pp. 218–225 (1983).

[49] Tamai, T.: A Class of Fixed-Point Problems on Networks and a Unified Algorithm, Research Report No. 89-07, Graduate School of Systems Management, the University of Tsukuba, Tokyo (1989).

[50] Tarjan, R.E.: Depth First Search and Linear Graph Algorithms, *SIAM J. Comput.*, Vol. 1, No. 2, pp. 146–160 (1972).

[51] Tarjan, R.E.: Testing Flow Graph Reducibility, *J. Comput. and Syst. Sci.*, Vol. 9, pp. 355–365 (1974).

[52] Tarjan, R.E.: Graph Theory and Gaussian Elimination, in Bunch, J.R. and Rose, D.J. eds.: *Sparse Matrix Computation*, Academic Press, New York, (1976).

[53] Tarjan, R.E.: Iterative Algorithms for Global Flow Analysis, in Traub, J.F.ed.: *Algorithms and Complexity, New Directions and Recent Results*, Academic Press, New York, pp. 91–102 (1976).

[54] Tarjan, R.E.: Applications of Path Compression on Balanced Trees, *J. ACM*, Vol. 26, pp. 690–715 (1979).

[55] Tarjan, R.E.: A Unified Approach to Path Problems, *J. ACM*, Vol. 28, pp. 577–593 (1981).

[56] Tarjan, R.E.: Fast Algorithms for Solving Path Problems, *J. ACM*, Vol. 28, pp. 594–614 (1981).

[57] Ueda, K.: Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard, ICOT Tech. Repr. TR-208, Institute for New Generation Computer Technology, Tokyo (1986).

[58] Ullman, J.D.: Fast Algorithms for the Elimination of Common Subexpressions, *Acta Inf.*, Vol. 2, pp. 191–213 (1973).

[59] Wegbreit, B.: The Synthesis of Loop Predicates, *Comm. ACM*, Vol. 17, No. 2, pp. 102–112 (1974).

[60] Wegbreit, B.: Property Extraction in Well-Founded Property Sets, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 3, pp. 270–285 (1975).

[61] Yannakakis, M.: Computing the Minimum Fill-in is NP-Complete, *SIAM J. Alg. Disc. Meth.*, Vol. 2, No. 1, pp. 77–79 (1981).

[62] ICOT 第四研究室：KL1 プログラミング　入門編／初級編／中級編，新世代コンピュータ技術開発機構 (1989).

[63] 伊理正夫，他：ネットワーク構造を有するオペレーションズ・リサーチ問題の電算機処理に関する研究，日本オペレーションズ・リサーチ学会，報文シリーズ T-73-1 (1973).

[64] 伊理正夫，韓太舜：線形代数 — 行列とその標準形，教育出版 (1977).

[65] 岩村聯：束論，共立出版 (1966).

[66] 岡本吉晴，玉井哲雄：ネットワーク計画システム，三菱総合研究所所報，No. 5，pp. 48–71 (1977).

[67] 久保田光一：高速自動微分法と応用，東京大学大学院工学系研究科学位論文 (1989).

[68] 後藤敏：ネットワークシステムの設計理論の研究，早稲田大学理工学部学位論文 (1977).

[69] 今野浩，鈴木久敏編：整数計画法と組合せ最適化，日科技連 (1982).

[70] 玉井哲雄：グラフ上の一群の不動点問題とその逐次型解法，電子情報通信学会論文誌 A，Vol. J76-A, No. 1，(1993).

[71] 玉井哲雄：グラフ上の不動点問題に対する消去型解法の構造，電子情報通信学会論文誌 A，掲載予定，(1993).

[72] 玉井哲雄：基底逆行列，反町洋一編：線形計画法の実際 第3章，産業図書，pp. 30–72 (1992).

[73] 玉井哲雄，三嶋良武，松田茂広：ソフトウェアのテスト技法，共立出版 (1988).

[74] 玉井哲雄，福永光一：記号実行システム，情報処理，Vol. 23, No. 1, pp. 18–28 (1982).

[75] 室田一雄：グラフの Menger 型分解，情報処理学会 論文誌，Vol. 23, No. 3, pp. 280–287 (1982).

[76] 森下真一：完備束値をもつ論理型言語，コンピュータソフトウェア，Vol. 6, No. 4, pp. 68–82 (1989).

[77] 米澤明憲，柴山悦哉，Briot, J.-P., 本田康晃，高田敏弘：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，Vol. 3, No. 3, pp. 189–203 (1986).