解　説

算　法　論　理†

沢　村　一††

1. まえがき

計算機のソフトウェアは何らかの言語で表現されたプログラムから成っている．そのプログラムを数学的対象としてとらえ解析しようとする理論が計算機科学の一つの重要な分野として広く認められ研究されてきた．プログラムの基礎論を目ざしたこのような研究対象領域は今日ではMcCarthyによれば計算の数学的理論 (Mathematical Theory of Computation)，Ershovによれば理論プログラミング (Theoretical Programming)，五十嵐によればプログラムの理論 (Theory of Programs) と呼ばれている．

プログラムの理論という言葉で具体的にどのような研究対象が包含されるかはあまり明らかではないが大体次のように分類される．

（1） プログラミング言語の意味論，

（2） プログラムの性質の形式的証明に関する証明論，

（3） プログラムの(半)自動構成法に関する構成論．

もちろんこれら三つの範疇の境界ははっきりと区別されるものではなくむしろ互いに関連，影響し合っている．

プログラミング言語の意味論はプログラミング言語の定義の本質的な部分であり，プログラムの性質の証明論と密接なかかわりをもっている．意味論の目的は構文規則によって規定された文字列が何を意味するかを形式的に定義することであり，意味をいかなる抽象化のレベルでとらえるかによって，すなわちその言語で書かれたプログラムに対する計算機の動作を直接記述してしまうものからプログラムの検証の基礎となるよう意図したものまで多くの方法が考えられている[1]．代表的なものとして，

a）属性文法 (attribute grammar)[2]，Wijngaarden文法[3]などの構文解析木と関連して意味の定義が与えられるもの，

b）プログラムを抽象的な対象（状態，述語など）の変換としてとらえるもの[4-7]

などを上げることができる．後者はさらに抽象化のレベルによってより具体的なものから，operational[4]，denotational[5]，axiomatic[6],[7]な方法に分けられる．また最近の意味論の方法論として相補的な定義 (complementary definitions) によるものがある[1],[8]．これはプログラミング言語の意味論として，言語の設計者，利用者そしてその言語を実現しようとする人達にも等しく適したユニバーサルな意味の記述法を与えることが困難なために，個々の目的に応じた意味の定義の集合を与え，それらの相互関係をモデル理論的に，すなわちより具体的な定義がより抽象的な定義の一つの解釈を与えるということを証明しようとするものである．この定義法によって異なった意味論が同一の言語の意味を定義しているという保証を与え，さらにプログラミング言語の種々の目的に適した意味の完全な記述を与えることを可能にしている．そしてこれはプログラミング言語の意味の諸相をプラグマティクな観点から与えるものにもなっている．

構成論はプログラムの理論の三つの範疇の中で，関数あるいはプログラムの存在性の問題，構成概念の未発展のゆえに最も形式化が難かしいものと考えられている．したがってこれまでいろいろの試みが成されているにも拘らずまだその困難点の所在の分析，試行錯誤の段階にとどまっている．与えられた要求を満たすプログラムを自動的にあるいは人と計算機との対話を通して作り出すことは人工知能研究における問題解決，特にプランニングの問題とも密接に関係しているが，Biermann[9]はこの方面の研究の方法論を次のように分類している．

a）伝統的自動プログラミングの拡張として高水準プログラムや特殊目的プログラムの生成，

b）例（計算列例，入出力例）からのプログラムの構成，

† Algorithmic Logics by Hajime SAWAMURA (Division of Information Engineering, Faculty of Engineering, Hokkaido University).

†† 北海道大学工学部情報工学専攻

c）形式的入出力仕様からの構成,

d）自然言語コマンドの変換としてのプログラムの構成,

e）知識（データベース）に基づく発見的プログラム構成.

c）の形式的入出力仕様からの構成論は(2)の証明論との結びつきが強いが，これまで大体次のようなアプローチが考えられている.

c1） 入出力仕様を1階述語論理式あるいは2階述語論理式で与え機械的証明手続き，導出原理を用いて証明しその証明図からプログラムを導くもの[10),11)]と直観主義論理の体系を用いるもの[12)]，さらに論理式の Gödel 解釈によるもの[13)],

c2） プログラムの構成要素に対する Hoare 型の意味論に基づき入出力仕様をサブゴールへ分解していくもの[14)].

プログラムの証明論はプログラムのもつ算法的な性質，例えば正当性 (correctness), 同値性 (equivalence), 停止性 (termination) などを命題として記述し，公理，推論規則を明確に定めることによって形式的な証明を与えることを目的としている．プログラムの性質を証明するために論理の言葉で表現することの利点はそれらを計算機によってかなり機械的に証明できるという点にある.

特にプログラムの正当性の証明はデバッギングに代るものとしてそして信頼性の高いプログラム作成に有効な手法として最も重要なものと考えられ，Floyd は1967年に「プログラムに意味を与えること」という論文の中でプログラムの正当性の概念を公理的方法により与えている[15)]．Floyd によるプログラムの検証法は今日帰納的表明法 (inductive assertion method) と呼ばれ，後にこの方法は Manna[16)]，Hoare[6)]によって形式化されたがその後のプログラムの検証法，さらにプログラミング方法論[17),18)]，プログラム言語の設計思想[19)]にも多大な影響を与えてきた.

プログラムの性質の証明法はそこで用いられる帰納的な方法によって分類することが最も分かり易いと思われる．これまで次の名で呼ばれている証明法が提案されている[20),21)]：リカージョン帰納法 (recursive induction), 構造帰納法 (structural induction), 帰納的表明法 (inductive assertion), 計算帰納法 (computational induction), 打切り帰納法 (truncation induction), 不動点帰納法 (fixpoint induction), サブゴール帰納法 (subgoal induction)[22)]．さらに証明の対象となっているプログラム形としてはフロチャート形，リカーシブ形，パラレルプログラム形を含み，プログラミング言語としては ALGOL, PASCAL の断片に加えて PL/I, APL, アセンブラ，マイクロプログラムにまで及んでいる．プログラムの種類としては自然数あるいは複雑なデータ構造を操作するプログラムから最近ではオペレーティングシステム，モニタープログラムの証明法にまで研究が進められている.

さてこれまでプログラムに関する証明論の進んできた道をかなり荒く辿ってきたがそこでの証明論では具体的なプログラミング言語で書かれたプログラムの世界とその性質を記述するための言語の世界がはっきり別々に存在しそれら二つの世界の間の関係を議論するという方法が取られていたように考えられる．算法論理 (algorithmic logic) とはこのような境界を意識せず一つの論理的枠組の中でアルゴリズムを構成する概念やその性質記述法を抽象的にとらえようとする立場から生まれてきた一つの形式的論理体系である（この意味では最近のデータの抽象化技法の考え方と類似している）．抽象化そのものの意義についてはここで述べるつもりはないが少なくとも同一の土俵の上でプログラムとその性質や証明を議論することはそれらに関する問題の理解を深め本質を知るための助けとなるであろう．特に前に記した意味論，構成論への新たな寄与となるに違いないと思われる．その抽象化の仕方を

a）アルゴリズムを構成する基本的概念,

b）アルゴリズムの性質を記述するのに必要な論理的概念

とは何かそしてそれらをどのように設定するかによっていろいろな算法論理が考えられている.

本稿では，最近の代表的な算法論理と考えられる，Hoare, Salwicki, Dijkstra, Pratt, Kröger, Constable の論理について解説し，ソフトウェア工学への理論的アプローチとしてそれに与える影響，寄与といったものについても言及する.

2. 算法論理とは

ここでは算法論理とは何かについてそれを直接的に規定することはしない．むしろ代表的研究者の算法論理を形式化するさいの考え方，動機などを通して算法論理とは何かについて述べる.

算法論理という言葉を最初に用いその後ポーランドの Salwicki, Rasiowa を中心とする算法論理の研究者に強い影響を与えた Engeler は算法論理研究の諸側面

に関して次のような点を上げている[23].

- アルゴリズムに関する基本的概念や問題（例えばプログラムの意味，正しさ，関数の存在の証明すなわちアルゴリズムを構成することなど）を研究するために適用される形式論理の方法を研究すること.
- 形式論理における諸方法の算法的研究を行うこと（例えば，決定手続きとそれらの複雑さ，命題の簡単化など），
- アルゴリズムを調べるために論理的方法の算法的研究を行うこと（例えば，プログラムの正当性の自動証明，自動プログラム構成など）など.

さらに，このような算法論理は次の主要な三つの要求を満たしているべきであると述べている，

- 停止性，同値性，部分正当性のようなプログラムの性質があいまいさなく表現できること，
- 形式的証明システムが得られうること，
- 形式的体系についてメタ理論的問題が取り扱えること，例えば計算機上での証明手続きの実現可能性の限界，他の論理との関係など.

また Salwicki, Rasiowa は次のように書いている[24),25)]. 一つの形式化された言語において，そのいくつかの表現はプログラムと解釈され他のものはプログラムの性質を記述する論理式と解釈されるような統一的な言語を導入することが第一の目的である. そしてそれによって計算過程についての法則を計算機械，プログラミング言語，計算の対象物とは独立に形式化し調べることを可能にし，さらにプログラミングの過程で起る問題についての方法論的研究あるいはプログラミングを改善していこうとする研究に役立つであろう.

Hoare[6]は計算機プログラミングを，プログラムのあらゆる性質やプログラムを実行して得られる帰結が純粋に演繹的な論証を通してそのプログラムテキストのみから得られるという意味で一つの厳密科学であると考えている. そして最初幾何学の研究において適用され後に数学の他の分野にも拡げられてきた公理的方法によって，言いかえると計算機プログラムについての我々の論証の基礎となる公理，推論規則を明らかにすることによって計算機プログラミングの論理的基礎を与えた.

Dijkstra は構造化プログラミング[17]の提唱者として知られているが，これまでのプログラムの作成とその正しさの証明という分業方式を批判しそれらは並行して行われるべきであると主張している[26]. そしてこの目的に合うアルゴリズム構成概念は何か，それらの意味はどのように与えられるかを示した[7),27)]. すなわち Dijkstra の論理は算法論理としての形式化以外にそのうちに一つのプログラミング方法論を考慮した算法論理であると考えることができる.

しかしながら，プログラムとその正しさの証明は同時に展開していくべきであるという Dijkstra の主張は具体的にはどのように行っていくのかという点ではあまり明確ではない. Constable はこの点に形式的に答え得るような一つの算法論理を提案している[28),29)]. Constable は Dijkstra や Engeler と同様にプログラムの停止性を最も基本的な性質と見なし実際的なプログラミングの論理であると共に形式的でもある論理を目ざしている.

Pratt による算法論理は動的な論理(dynamic logic)と呼ばれているがこれは Floyd および Hoare の論理の意味論的考察から得られたものである[30]. すなわち Floyd-Hoare の論理では“入力条件を満たしプログラムが停止するならば出力条件を満たす”というようなプログラムの性質記述の仕方を言語の基本的な構成単位としているのに対し，Pratt の論理では，我々は日常プログラムについて語るとき“プログラムを実行した後で……”のような言い方をするという観察から入力，出力条件を分離し，“プログラムの実行後出力条件が成立する”というように，プログラムを自然言語の文法のカテゴリーで言えば一種の副詞のようにとらえた表現を基本的言語要素とする算法論理を考えている. そしてこのような表現を記述するために様相論理 (modal logic) からの表現と意味論を取り入れている.

Kröger の論理はちょうど組合せ論理 (combinatory logic) が通常のいろいろな論理の前論理 (prelogic) 的な性格を有しているようにこれまで提案されてきた算法論理の前論理的性格をもっているように思われる. 実際 Kröger はアルゴリズムを連結したり，分岐させたり，くり返したりすることを思考の法則と考え，したがってこれらをそれぞれ命題の連結，分岐，くり返しを意味する論理的概念であるととらえている[31]. その結果異なった種類のプログラムの正当性の証明などは Kröger の論理では一様に取り扱うことが可能となっている[32),33)].

最後に Boas の，算法論理に対する要求として次のものを上げておこう[34]. これは先に記した Engeler の

算法論理に付加されるべき三つの要求にさらに加えられるべきものとして注目に値する.

- 形式的体系は構文的および意味的合成性 (compositionality) の条件 (Frege の原理といわれる) を満たしていること, すなわち複合文 (式) の形体とその意味はその構成要素のそれらからのみ合成されるべきであること.

この要求が算法論理を考える上で意味する点については次の章の終りで触れることにする.

これまで算法論理研究の動機, 基本的考え方, 算法論理によって何を意図しているかなどを見てきた. 次の章ではこれまでの算法論理に含まれる諸概念と算法論理間の技法的相違を見てみよう.

3. 代表的算法論理の概観

以下で述べる各算法論理がアルゴリズムをいかなる表現で記述しているかを明確にするために簡単なプログラミング言語を導入する.

3.1 簡単なプログラミング言語*

記号 1) 個体の上を動く変数記号, 2) 定数記号, 3) 算術記号, 4) 等号 $=$, 5) 論理記号 $\neg$, $\vee$, $\wedge$, $\supset$, $\forall$, $\exists$, 6) プログラム記号 ;, $:=$, if, then, else, fi, while, do, od.

項とは変数記号, 定数記号, 算術記号から構成される算術式を意味し, 論理式とは項, 等号記号, 論理記号, 変数記号から作られる記号列を意味するものとする.

プログラム

1) x が変数で t が項のとき $x:=t$ はプログラムである (代入文),
2) S_1, S_2 がプログラムのとき S_1; S_2 はプログラムである (合成文),
3) S_1, S_2 がプログラムで p が論理式のとき if p then S_1 else S_2 fi はプログラムである (条件文),
4) S がプログラムで p が論理式のとき while p do S od はプログラムである (くり返し文),
5) 1) から 4) で作られるもののみがプログラムである.

このプログラミング言語の意味は直観的に理解されうると思われるのでその意味の定義は省略する. 以下で年代順に Hoare, Salwicki, Dijkstra, Pratt, Kröger, Constable の算法論理についてその概略を非形式的に主として次の 4 点に焦点を当てて述べていく,

(1) 既存の論理との関係,
(2) 算法論理におけるアルゴリズムの表現形式,
(3) 算法論理に含まれる論理的概念の直観的意味,
(4) プログラムの性質記述法, 特に正当性の表現法.

したがって各算法論理の言語, 意味論, メタ的性質などの詳細は含まれない. また, この章の最後で内包論理に基づく算法論理についても触れることにする.

3.2 Hoare Logic

(1) Hoare[6]は Floyd[15]によるフローチャート形プログラムの正当性を証明するための帰納的表明法をプログラムテキストの証明を行えるようにループの証明に必要なループ不変式 (loop invariant) の存在を仮定し, Gentzen 型の証明システムを与えた. そこではプログラムの入力, 出力条件, プログラム変数間に成り立つ条件式などはすべて一階述語論理の論理式で記述されているが, プログラムは個体を操作するものであり, 他方一階述語論理が個体間の関係の論理であることからプログラムの性質は一階述語論理を基礎とすることによって記述され証明されるということが理解されるであろう.

(2), (3) Hoare の論理におけるアルゴリズム表現形式, 論理的概念はそれぞれ具体的なプログラミング言語であり, 例えば PASCAL[35], 一階述語論理のそれである. ただしループを実行している間つねに成り立っているループ不変式は一階述語論理の範囲内でいつも存在するとは限らない. その意味で Hoare の論理は不完全である[36]. しかしながら Cook は部分正当性のための Hoare 風の公理系に対して相対的完全性 (relative completeness) という概念を導入し, いくつかの特定の領域に対してはループ不変式がいつも存在することを示した[37].

(4) Hoare はプログラムの部分正当性を表現するのに $P\{S\}Q$ なる記法を導入した. これは "述語 P がプログラム S の実行前に成立していれば, それが正常に停止するならば, 実行後述語 Q が成立する" ということを意味する.

Hoare の論理はこれまで理論的にも実際的にも最もさかんに研究されてきた論理であり, 計算機科学の関連する他の領域に与えてきた影響は大きい. 実際的には, 例えば Hoare の論理に基づいた検証システムが作成されており[38], またつねに検証ということを意識

* この定義は不正確である. しかしながらその意味するところは推察されよう.

してプログラミングの方法を解説することを意図したものなど[39]がある．理論的には（3）でも言及したように Hoare の論理のメタ的性質を調べているもの，数々の拡張とヴァリエーションを追求するもの[40)~42)]などがある．

3.3 Salwicki Logic

（1） Engeler[43]はプログラムの各命令に番号をつけることによってプログラムのあらゆる可能な制御の流れを正則表現で表わし，正則表現に無限長論理 (infinitary logic) $L_{\omega_1,\omega}$ の論理式を対応させる方法を示すことによってプログラムの性質を記述した．そしてプログラムが停止することと対応する論理式が正しいこととは同値であることを証明した．他方Salwicki[24]は（2）で述べるような FS 式と呼ばれるプログラムを導入しこれに Salwicki の論理における論理式を対応させ Engeler と同様の定理を証明した．さらに Mirkowska[44]は Salwicki の言語を拡張し無限長論理に基づく Hilbert, Gentzen 型の完全な公理化を与えた．

（2） FS 式は次のものから成っている，

$[x_1/t_1,\ x_2/t_2,\ \cdots,\ x_n/t_n]$（同時代入文），

$\circ[S_1S_2]$（合成文），

$\curlyvee[pS_1S_2]$（条件文），

$*[pS]$（くり返し文），

$[p]$ p は開論理式．

ここで，記号 $\circ$, $\curlyvee$, $*$ はそれぞれ **3.1** のプログラムの合成文，条件文，くり返し文中に現われるプログラム記号に対応している．また開論理式とは限量記号をまったく含まない論理式を意味しこのとき，$[p]$ は一種のガードの働きをする．すなわちこのプログラム構成要素は Pratt の論理にも含まれ，Dijkstra の論理のガードをもつコマンドのガードだけをプログラムステートメントと見なしたものに相当する．

（3） Salwicki の論理式は開論理式に加えて，K を FS 式，α を論理式とするとき，$K\alpha$, $\cup K\alpha$, $\cap K\alpha$ も論理式として定義されそれらの直観的意味はそれぞれ，

$K\alpha$ は "K を実行した後で α"，

$\cup K\alpha$ は無限論理和 "$\alpha\vee K\alpha\vee KK\alpha\vee\cdots$"，

$\cap K\alpha$ は無限論理積 "$\alpha\wedge K\alpha\wedge KK\alpha\wedge\cdots$"

と解釈される．論理式 $*[pS]\alpha$ はこの $\cup$ を用いて同値式

$*[pS]\alpha\equiv\cup\curlyvee[pS[\ ]]$ $(\neg p\wedge\alpha)$

によって取り扱われている．

（4） Salwicki の論理では α, β を論理式，T を恒真な命題定数，K をプログラムとするとき，$(\alpha\Rightarrow K\beta)$ は強正当性を，$((\alpha\wedge KT)\Rightarrow K\beta)$ は弱正当性を意味する命題を表わす．

Salwickiの論理の種々の拡張，メタ的性質の研究，Hoare の論理との関係などについては Banachowski 他[45]にまとめられている．

3.4 Dijkstra Logic

（1），（3） Dijkstra[7),27)]はプログラムすべき問題の構造を損わずに自然な形でプログラミングができるアルゴリズム構成概念として非決定性 (nondeterministic) プログラミング言語を導入しさらにその意味を定義するために述語変換子 (predicate transformer) wp を導入した．この wp を使うとプログラム S と S の後件式 (postcondition) Q が与えられたとき $wp(S,Q)$ は S の実行が正常に停止し S の終了状態が後件式 Q を満たすような初期状態を規定する条件式のうちで最も弱い条件を表わす述語—最弱前件式 (weakest precondition)—となる．Dijkstra は非決定性プログラムの各構成要素 S と後件論理式 Q に対して $wp(S,Q)$ の作り方を構成的に与えたがくり返し文に対する $wp(S,Q)$ は一般には無限長論理式となる．この意味でループのとり扱いは Engeler, Salwicki と似ているが Dijkstra は Hoare と同様，逆向きに (backward) 述語を作っていく方法をとっている．

（2） 非決定性プログラミング言語は次のものから成る，

$x:=t$（代入文），

S_1; S_2（合成文），

if $B_1\to S_1[]\cdots[]B_n\to S_n$ fi（非決定性条件文），

do $B_1\to S_1[]\cdots[]B_n\to S_n$ od（非決定性くり返し文）

skip, abort.

（4） Dijkstra の論理では S をプログラム，P,Q を一階述語論理の論理式，Tを恒真な命題定数とすれば，入力述語 P に関してプログラム S が停止することは $P\supset wp(S,T)$，S が入出力述語 P，Q に関して強正当であることは $P\supset wp(S,Q)$ と表わされる．

述語変換子の論理的性質は文献 7), 27)で，Dijkstra の論理に基づく決定性プログラムの強正当性の検証法は文献 46) で議論されている．

3.5 Pratt Logic

（1） Pratt は様相論理に基づいてプログラムに対する論理的枠組を築いた[30]．その基本的アイディアは

他の算法論理とはまったく異なりプログラムを様相記号（あるいは副詞）と見なすというものであった．例えばαをプログラム（一般には非決定性プログラム），pを論理式とするとき新しい論理式$\langle\alpha\rangle p$が作られ，これは真偽論的様相論理 (alethic modal logic) の可能命題 $\Diamond p$（"pが成立することは可能である"を意味する）に対応する．またこれに双対な必然性を表わす様相記号$\Box$に対応するものは

$$[\alpha]p \equiv \neg\langle\alpha\rangle\neg p$$

によって定義される．

（2） Pratt の論理では Dijkstra の論理と同様非決定性プログラミング言語を対象としていて，次の構成要素から成っている，

$x:=t$（代入文），

$p?$（テスト文），

$\alpha|\beta$（非決定性分岐文），

$\alpha;\beta$（合成文），

α^*（非決定性くり返し文），

擬プログラミング言語要素（これはプログラムの性質の記述にのみ用いられる），

$x:=?$（ランダム代入文），

α^-（αの逆向き実行文）．

ここでα^-はプログラムαを逆方向に実行するプログラムを表わす．これらを用いると通常の条件文，くり返し文はそれぞれ，

$p?;\ S_1|\neg p?;\ S_2$,

$(p?;\ S)^*;\ \neg p?$

と書かれる．pが論理式であるとき，$p?$の働きはpが真のとき計算はpを通過することができるがそうでなければ計算はそこでブロックされるというものである．

（3），（4） Pratt の論理ではプログラムαの意味はプログラムの関係意味論 (relational semantics)[8]から，論理式は Kripke の可能的多世界意味論(possible world semantics)[47]に基づく様相論理の意味論にそって与えられている．これによると$[\alpha]p$の直観的意味は"αが停止するときはいつでもpが成立する"，$\langle\alpha\rangle p$は"αが停止しそのときpが成立する"となる．したがってプログラムの正当性は Hoare の $p\{\alpha\}q$ に対応するものは $p\supset[\alpha]q$，プログラムの停止性は true を恒真な命題定数とすれば $p\supset\langle\alpha\rangle$ true と表現される．

3.6 Kröger Logic

（1） Kröger の算法論理の究極の目的は我々がプログラムについて非形式的に論証するときの自然な仕方を形式化することであった[31)~33)]．この目的を実現するために Kröger は時間軸上での論証を可能にする位相的論理 (topological logic)[48]と言われている一種の時間（制）の論理 (temporal or tense logic) とループ概念に関する自然な論証を遂行できるように完全な無限長論理の公理系を導入した．

（3） Kröger の論理の論理式は ω^*（ωは最小の超限順序数）までの順序数の集合を時間軸とし，その任意の時点で解釈される．また次のような特別の論理的概念を含んでいる，

△ "…そしてその次に…"，

○ "…をくり返す"，

$\Box^\alpha$ "α 時間後…"，

$\curvearrowright$ "もし…ならばその後で…"，

I^α "次の α 時間の間は何も変化しない"．

A を Kröger の論理式とし，$\tau(A)$ を任意の時点 α で A の真偽値を評価するのに必要な時間を表わす順序数とするとき，上の論理的概念間の関係のいくつかは次のようになっている，

$$A\triangle B \equiv A\wedge\Box^{\tau(A)}B,$$

$$A\curvearrowright B \equiv A\supset\Box^{\tau(A)}B \equiv \neg(A\triangle\neg B).$$

（2） ここでは **3.1** の簡単なプログラミング言語を Kröger の論理の論理式に関連づける表現関数 R を用いて書くと次のようになる，

$R(x:=t)=[x:=t]$（代入文），

$R(S_1;\ S_2)=R(S_1)\triangle R(S_2)$（合成文），

R (if then S_1 else S_2 fi)（条件文）

$$=\begin{cases} p\wedge R(S_1)\vee\neg p\wedge R(S_2), \\ \tau(R(S_1))=\tau(R(S_2)) \text{ のとき}, \\ p\wedge R(S_1)\triangle I^\alpha\vee\neg p\wedge R(S_2), \\ \tau(R(S_1))+\alpha=\tau(R(S_2)) \text{ のとき}, \\ p\wedge R(S_1)\vee\neg p\wedge R(S_2)\triangle I^\alpha, \\ \tau(R(S_1))=\tau(R(S_2))+\alpha \text{ のとき}, \end{cases}$$

R(while p do S od)$=\circ(p\wedge R(S))\triangle(\neg p\wedge I^1)$

（くり返し文）．

（4） Hoare の論理の弱正当性命題 $P\{S\}Q$ は Kröger の論理では $P\wedge R(S)\curvearrowright Q$ と表わされる．

3.7 Constable Logic

（1） Constable は命題論理に相当する単項プログラミング論理(monadic quantifier-free programming logic)，状態の集合上の限量を許す単項プログラミング論理，述語論理に相当する多項プログラミング論理 (polyadic) を順次構成した[28)]．結果として，αをプロ

グラム，p を述語とするとき $\alpha; p$ が基本命題，プログラムが決定性であるという点で Salwicki の論理に似た算法論理となっている．

（2） Constable のプログラミング言語は次のものから成っている，

$x:=t$（代入文），

$\alpha; \beta$（合成文），

$(p\rightarrow S_1, S_2)$（条件文），

$p^*\alpha$（くり返し文）．

（3） 論理式は一階述語論理式，プログラム，プログラム α と論理式 p に対して $\alpha; p$ から帰納的に定義され，論理式としてのプログラム α は "α は停止する"，$\alpha; p$ は "α が停止し p が成立する" を意味する．したがって $p; \alpha$ は "p が成り立ち α が停止する" を意味している．このようにプログラムそのものが命題と解釈されることは 2. でも述べたようにプログラムとその正しさの証明は同時に行われるべきであるというプログラミングの方法論に関する Dijkstra のテーゼの一つの具現化であると考えられる．

（4） Constable の論理式 $\alpha; p$ は ILL[38] のシステムでの α; assert p, Dijkstra の $wp(\alpha, p)$ と内容的に同じ働きをし，Hoare の $p\{S\}q$ に相当する命題は $p\Rightarrow(S\Rightarrow S; q)$，停止性は $p\Rightarrow S; q$ と表わされる．

Constable の上記のような算法論理の考えに基づいた大変興味深い実際的な，PL/CV と呼ばれるプログラミング論理に関する単行本が出版されている[49]．

3.8 内包論理に基づく算法論理

一つの形式化された論理体系の中で特別の言語的現象を扱おうとするときしばしば言語の内包性（intensionality）が問題になり数学的難点に落ち入ることがある．言語学，論理学の歴史においてはこれまで言語の内包性に関する多くの議論があった．言語としてのプログラミング言語においても内包性の問題が生ずることが Janssen と Boas により指摘されている[50),51]．次のような自然言語とプログラミング言語の例を見てみよう．

$$\frac{\text{The temperature is ninety}\quad \text{The temperature rises}}{\text{ninety rises}} \qquad (1)$$

$$x=x+1\{p:=x;\ x:=x+1\}p=x \qquad (2)$$

$$1=a(j)\{a(i):=1\}a(i)=a(j) \qquad (3)$$

ここで x は単純変数，a は配列変数，p はポインター変数である．（1）は奇妙な推論であり，（2），（3）は Hoare の論理の代入文に関する公理図式 $P(t)\{x:=t\}P(x)$（$P(t)$ は $P(x)$ における自由変数 x を項 t で置き換えて得られる）の具象例であるが，（2）は矛盾した命題，（3）は不十分な命題（前件条件は $i=j \vee 1=a(j)$ となるべきである）となっている．これらの不都合さはすべて表現を構成している構成要素を外延性を仮定し単純に他の表現で置き換えたことから生じている．Montague は内包的な表現，限量化表現などを含む英語の断片の論理分析を行う上で言語表現の外延(extension)，内包（intension）を明確に区別する高階の様相論理である内包論理（intensional logic）を形式化し（1）のような問題を解決した[52]．Janssen と Boas はこの内包論理を用いて（2），（3）のような代入文の意味に関する内包性の問題を解決した[50),51]．その基本的なアイディアはさまざまな変数の外延，内包を区別する表現を用いることであった．変数の外延とは可能な世界（計算機の可能な内部状態）を指定したときの変数の取る値であり，内包とは可能な世界をその世界における変数の値に写像する関数である．したがって例えばポインター変数 p の外延 $^{\vee}p$ は特定の状態でそれが指す変数（あるいはアドレス）であり，内包 $^{\wedge}p$ は状態の集合 S から変数の集合 V への関数となる．このように変数概念の外延，内包という2面性によってあらゆる型をもつ代入文の意味を内包論理の枠組の中で一様に把握することができる．内包論理に基づく算法論理としての形式化の試みは 文献 53）で行われている．さらに内包論理は今後内包性が問題になる自然言語の計算機による処理，データの意味論を考えていく上で重要なものとなろう．

これまでいろいろな算法論理を概観してきた．これら算法論理間の形式的関係を明らかにすることは理論的にも，実際的にも算法論理のいろいろな面を評価する上で重要であると思われる．このような方向の試みは文献54）で行われている．また Pratt の論理と他の算法論理との関係は Harel[55] によって議論されている．

4. 算法論理のソフトウェア工学への寄与

ソフトウェア工学とは品質の良いソフトウェアを能率よく開発し保守するための技術の総称であると言われ，その技術体系はもちろんすでに完成しているわけではなくこれまでの現場からの経験によって生まれてきた技術とか，計算機科学におけるさまざまな問題の理論的，実際的追求から生まれてきた知見，技術などがそれを形成していくであろう．これまでのソフトウ

ェア工学としての問題意識の中には，ソフトウェアの信頼性を高めるための仕様，検証法の開発が最も重要なものとして含まれていた[56]．

そして，これまで概観してきたように算法論理の研究がアルゴリズムを構成している抽象概念とその性質の記述法，それに基づく証明法が中心テーマであることから，算法論理研究に基づくソフトウェアの仕様技術，検証技術の開発は今後のソフトウェアの信頼性技術の発展に基礎的な役割を演じ重要な寄与を与えていくことになると思われる．また算法論理に基づいたプログラムの構成法の研究も人工知能，データベース論などで得られた技術と共にソフトウェア工学を支えるツールのための基礎として重要なものとなろう[57]．

最後に，算法論理の自然言語理解への興味ある応用として Pratt[58]の例を上げておこう．我々は“もしテレビが映らなければそれを蹴飛ばしてもそれは依然として映らないであろう”という自然文の妥当性から“もしテレビが映らなければ何度それを蹴飛ばしてもそれは依然として映らないであろう”という文の妥当性を Pratt の論理の推論規則

$$\frac{p \supset [a]p}{p \supset [a^*]p}$$

を用いて容易に導くことができる．このような応用はアクション(action)による状態の変化についての推論の実現に有益である．

5. あ と が き

これまで多くのいろいろな特徴をもつ算法論理が提案され，理論的には算法論理としてのいろいろな形態とその在り方に興味深いものがあった．しかしながら本稿ではこれまで知られている算法論理をすべて網羅しているわけではない，ここで述べることができなかったものに Ashcroft の Lucid[59]などがある．他方各算法論理研究者のさまざまな設計思想をどのように判断し我々の目的に最も適した算法論理をどのように選択していったらよいであろうか？ これまでのいろいろな算法論理の技術的相違については 3. で概観した通りであるがさらに大切なことはそれら算法論理に対するプルーフチェッカーなどと共に多くの実際的経験を得，それらの実際的適用可能性を比較することであろう．そうすることによって上の問とさらに今後の算法論理の展開はどうあるべきかに対する解答を得ることができると思われる．筆者は内包論理による算法論理も含めてこれらの算法論理を計算機上で実現し評価してみたいと思っている．

終りに，論理学がデータベース論に対しても重要であることが最近認識されてきているが[60]，データベース論に現われる諸概念，実体(entity)，範疇(category)，名前(name)，関係(relationship)，属性(attribute)など[61]といくつかの算法論理の基礎となっている様相論理，あるいは哲学的論理学におけるそのような概念[62),63]との関係を考察することは算法論理を一つの論理形態としてデータベース論へ応用していくことを考える上ではもちろん，さらに今後計算機科学の哲学的基礎を確立する上でも意義があるように思う．

謝辞 本稿を書くきっかけを与えて下さった電子技術総合研究所間野暢興氏と日ごろ御討論いただいている北大情報工学専攻宮本衛一助教授，桃内佳雄助手，同工業数学講座前田隆助手に感謝する．

参 考 文 献

1) Donahue, J.E.: Complementary definitions of programming language semantics, Lect. Notes in Comp. Sci. 42, p. 172, Springer-Verlag (1976).
2) Knuth, D.E.: Semantics of context-free languages, Mathematical Systems Theory, Vol. 2, No. 2, pp. 127-145 (1968).
3) Wijngaarden, A. et al.: Revised report on the algorithmic language ALGOL 68, Acta Infor., Vol. 5, pp. 1-269 (1975).
4) Wegner, P.: The Vienna definition language, Computing Surveys, Vol. 4, No. 1, pp. 5-63 (1972).
5) Stoy, J.E.: Denotational semantics: The Scott-Strachey approach to programming language theory, p. 414, The MIT Press (1977).
6) Hoare, C. A. R.: An axiomatic basis for computer programming, CACM, Vol. 12, No. 10, pp. 576-583 (1969).
7) Dijkstra, E.W.: A discipline of programming, p. 217, Prentice-Hall, Inc. (1976).
8) Hoare, C.A.R. and Lauer, P. E.: Consistent and complementary formal theories of the semantics of programming languages, Acta Infor., Vol. 3, No. 2, pp. 135-154 (1974).
9) Biermann, A. W.: Approaches to automatic programming, in Advances in computers, Vol. 15, pp. 1-63, Academic Press (1977).
10) Waldinger, R. J. and Lee, R. C. T.: PROW: A step towards automatic program writing, lst Int. Jt. Conf. Artif. Intel., pp. 241-252 (1969).
11) Darlington, J.L.: Automatic program synthesis

in second-order logic, 3rd Int. Jt. Conf. Artif. Intel., pp. 537-542 (1973).
12) Miglioli, P. A. and Ornaghi, M.: A calculus to build up correct programs, Lect. Notes in Comp. Sci., MFCS, pp. 398-409 (1977).
13) Goto, S.: Program synthesis through Gödel's interpretation, Proc. of the International Conf. on Mathematical Studies of Information processing, pp. 287-306 (1978).
14) Buchanan, J.R.: A study in automatic programming, Stanford Artificial Intelligence Laboratory, MEMO AIM-245 (1974).
15) Floyd, R.W.: Assigning meanings to programs, Proc. Amer. Math. Soc. Symposia in Applied Mathematics, Vol. 19, pp. 19-31 (1967).
16) Manna, Z.: The correctness of programs, JCSS, Vol. 3, No. 2, pp. 119-127 (1969).
17) Dahl, O. J., Dijkstra, E. W. and Hoare, C. A. R.: Structured programming, Academic press (1972), 野下・川合・武市共訳: 構造化プログラミング, p. 249, サイエンス社 (1975).
18) 鳥井, 二木, 真野: プログラミング方法論の展望, 情報処理, Vol. 20, No. 1, pp. 22-43 (1979).
19) London, R. L.: Perspectices on program verification, in Currrent trends in programming methdology, Vol. 2, Program validation, pp. 151-172 (1977).
20) Manna, Z., Ness, S. and Vuillemin, J.: Inductive methods for proving properties of programs, CACM, Vol. 16, No. 8, pp. 491-502 (1973).
21) Manna, Z.: Mathematical theory of computation, McGraw-Hill (1974), 五十嵐訳: プログラムの理論, p. 478, 日本コンピューター協会 (1974).
22) Morris, Jr, J. M. and Wegbreit, B.: Subgoal induction, CACM, Vol. 20, No. 4, pp. 209-222 (1977).
23) Engeler, E: Algorithmic logic, in Foundations of Computer Science, ed. J. W. deBakker, Math. Centre Tracts 63, Amsterdam, pp. 57-85 (1975).
24) Salwicki, A: Formalized algorithmic languages, Bull. Acad. Pol. Sci. Ser. Math. Astr. phys. 18, pp. 227-232 (1970).
25) Rasiowa, H.: Algorithmic logic, ICS PAS Reports 281, Warszawa (1977).
26) Dijkstra, E. W.: Programming: From craft to scientific discipline, 木村・和田訳: プログラミング—工芸から科学へ, 情報処理, Vol. 18, No. 12, pp. 1248-1256 (1977).
27) Dijkstra, E. W.: Guarded commands, nondeterminancy and formal derivation of programs, CACM, Vol. 18, No. 8, pp. 453-457 (1975).
28) Constable, R. L.: On the theory of programming logics, 9th Annual ACM Symp. on theory of computing, pp. 269-285 (1977).
29) Constable, R.L.: A constructive programming logic, IFIP 77, pp. 733-738 (1977).
30) Pratt, V. R.: Semantical considerations on Floyd-Hoare logic, 17th IEEE Symp. on Found. of Comp. Sci., pp. 109-121 (1976).
31) Kröger, F.: An extension of propositional logic by algorithmic concepts Abteilung Mathematik der TUM, Bericht Nr. 7419 (1974).
32) Kröger, F.: LAR: A logic of algorithmic reasoning, Acta Infor., Vol. 8, Fasc. 3, pp. 243-266 (1977).
33) Kröger, F.: A uniform logical basis for the description, specification and verification of programs, in Formal descriptions of programming concepts, E. J. Neuhold (ed.), pp. 441-459, North-Holland Pub. Comp. (1978).
34) Boas, P. van Emde: The connection between modal logic and algorithmic logics, Lect, Notes in Comp. Sci. 64, pp. 1-18, Springer-Verlag (1978).
35) Hoare, C. A.R. and Wirth, N.: An axiomatic definition of the programming language PASCAL, Acta Infor., Vol. 2, pp. 335-355 (1973).
36) Wand, M.: A new incompleteness results for Hoare's system, JACM, Vol. 25, No. 1, pp. 168-175 (1978).
37) Cook, S. A.: Soundness and completeness of an axiom system for program verification, SIAM J. on Comp., Vol. 7, No. 1, pp. 70-90 (1978).
38) Igarashi, S., London, R. L. and Luckham, D. C.: Automatic program verification I: logical basis and its implementation, Acta Infor., Vol. 4, pp. 145-182 (1975).
39) Alagic, S. and Arbib, M. A.: The design of well-structured and correct programs, p. 292, Springer-Verlag (1978).
40) 林・五十嵐: プログラムの理論, 情報処理, Vol. 17, No. 5, pp. 437-447 (1976).
41) Sokolowski, S.: Axioms for total correctness, Acta Infor., Vol. 9, pp. 61-71 (1977).
42) Owicki, S. and Gries, D.: An axiomatic proof technique for parallel programs I, Acta Infor., Vol. 6, pp. 319-340 (1976).
43) Engeler, E.: Algorithmic properties of structures, Mathematical Systems Theory, Vol. 1, No. 3, pp. 183-195 (1967).
44) Mirkowska, G.: Algorithmic logic and its applications in the theory of programs I, Fundamenta Informaticae, Vol. 1, No. 1, pp. 1-

17 (1977), II, ibid, Vol. 1, No. 2, pp. 147–165 (1977).
45) Banachowski, L., Krecmar, A., Mirkowska, G., Rasiowa, H. and Salwicki, A.: An introduction to algorithmic logic: Mathematical investigations into theory of programs, in Mazurkiewicz and Pawlak (eds.), Math. Found. of Comp. Sci., pp. 7-99, Banach Center Publications, Vol. 2, Warsaw (1977).
46) Basu, S. K. and Yeh, R. T.: Strong verification of programs, IEEE Trans. on Software Engineering, Vol. SE-1, No. 3, pp. 339-346 (1975).
47) Hughes, G. E. and Cresswell, M. J.: An introduction to modal logic, p. 388, Methuen and Co. Ltd. (1968).
48) Rescher, N. and Garson, J.: Topological logic, J. of Symbolic Logic, Vol. 33, pp. 537-548 (1968).
49) Constable, R. L. and O'Donnell, M.J.: A programming logic with an introduction to the PL/CV verifier, p. 389, Winthrop Pub., Inc. (1978).
50) Janssen, T. M. W. and Boas, P. van Emde: On the proper treatement of referencing, dereferencing and assignment, Lect. Notes in Comp. Sci. 52 (1977).
51) Janssen, T. M. W. and Boas P. van Emde: The expressive power of intensional logic in the semantics of programming languages, Lect. Notes in Comp. Sci. 53, pp. 303-311 (1977).
52) Montague, R.: The proper treatement of quantification in ordinary English, in R. H. Tomason (ed.) Formal philosophy-selected papers of R. Montague (1977).
53) Sawamura, H., Momouchi, Y. and Maeda, T.: The intensional logic of programs, manuscript (1979).
54) 沢村・前田・桃内: Algorithmic logicの比較研究, 電子通信学会, オートマトンと言語研究会 AL 78-27 (1978).
55) Harel, D.: Proving the correctness of regular deterministic programs: A unifying survey using dynamic logic, preprint (1979).
56) 国井監修: ソフトウェア工学, bit 8 (1978).
57) 謝 章文: プログラミングの形式化とソフトウェア・ツールのありかた, ソフトウェア工学シンポジウム, 情報処理学会, pp. 3-15 (1979).
58) Pratt, V. R.: Six lectures on dynamic logic, MIT/LCS/TM-117 (1978).
59) Ashcroft, E. A. and Wadge, W. W.: Lucid-A formal system for writing and proving programs, SIAM J. on Computing, Vol. 5, No. 3, pp. 336-354 (1976).
60) Gallaire, H. and Minker, J. (eds.): Logic and data bases, p. 457, Plenum Press (1978).
61) Kent, W.: Data and reality, p. 211, North-Holland Pub. Comp. (1978).
62) 野本和幸: 様相論理のモデル論と哲学的諸問題-S. クリプキの場合, 理想 1月 (1977).
63) 山本 巍: 問題としての実体一アリストテレス実体論の見取図一, 理想 5月 (1976).

(昭和 54 年 5 月 14 日受付)