The University of Osaka
Institutional Knowledge Archive

Title	代数的仕様記述の検証に関する研究
Author(s)	東野, 輝夫
Citation	大阪大学, 1984, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/158
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

代数的仕様記述の検証に関する研究

昭和59年2月

東野　輝夫

内 容 梗 概

本論文は，筆者が大阪大学大学院基礎工学研究科（物理系情報工学分野専攻）に在学中，嵩研究室において行った情報処理に関する研究のうち，代数的仕様記述の検証に関する研究を２章にまとめたものである．

緒論および各章の第１節では，研究の現状，その工学上の意義，本研究の新しい成果について概説している．又，各章の最後の節および全体の結論では，本研究で得られた主な結果と今後に残された問題点について述べている．

第１章では，まず筆者らが定義した代数的記述言語ASL/V による仕様の記述，並びに記述された仕様上での検証を支援することを主な目的とする代数的仕様検証支援系の概要について述べている．一般に，項の書き換えによって代数的仕様の検証を行う場合，検証に必要な性質については，たとえそれが基本的な数学の性質であっても，すべて検証者が公理の形で導入しなければならないが，本支援系では，ブール代数や整数の加減算，大小比較に関する性質を用いて検証を行う場合については検証が機械的に行えるよう，プレスブルガー文の真偽判定機能を組み込んでいる．第１章の後半ではクイックソートプログラムの検証を例に，この支援系の有効性について述べている．

第２章では，代数的に記述されたハイレベルデータリンク制御（HDLC）手順の検証について述べている．筆者らは，まずHDLC手順を用いて通信を行う通信系を抽象的な順序機械とみなしてその代数的記述を作成し，次にこの記述が公理系として”矛盾なく”且つ”不足なく”書かれていることを示すと共に，この記述の上で成立する幾つかの論理関係式（例えば，通信系全体の動作可能性や１，２次局の状態変数間に成立する関係等）の証明を行っている．これらの検証は主として構造的帰納法を用いて行い，検証作業には上述の検証支援系を利用した．

代数的仕様記述の検証に関する研究

目　　次

緒論 …… 3

第１章　代数的仕様検証支援系 …… 7

１．１　序言 …… 7

１．２　諸定義 …… 8

１．３　支援系の構成 …… １１

１．４　クイックソートプログラムの検証 …… １８

１．５　結言 …… ３０

第２章　代数的に記述されたHDLCプロトコルの検証 …… ３１

２．１　序言 …… ３１

２．２　諸定義 …… ３２

２．３　通信系全体の代数的記述 …… ３４

２．４　通信系全体の代数的記述の性質 …… ４７

２．５　通信系全体の公理系上での検証 …… ５０

２．６　検証の具体例 …… ５２

２．７　結言 …… ５６

結論 …… ５７

謝辞 …… ５８

文献 …… ５９

緒　　論

信頼性の高いソフトウェアを作成するためには，仕様を的確に記述することが不可欠であり，形式的な仕様記述法が幾つか提案されている．仕様を形式的に記述することにより，仕様のレベルで無矛盾性や完全性等のチェックが行い易く，仕様に従って作成されたプログラムの正しさを厳密に検証するための基礎が確立する．又，仕様のあい昧性をなくすことが出来るため，意志疎通用の媒体としても有効であると考えられる．

代数的仕様記述法は，抽象データ型の形式的な仕様記述法の１つであり，データ型を特徴付ける幾つかの演算の間に成立する関係を公理として記述することにより，そのデータ型の振舞を定義するものであり，意味の定義が簡潔で，書き手が選んだ任意の抽象化レベルで仕様を記述出来るなどの特徴を持つ．このため，抽象的な仕様から具体的な関数型プログラミング言語のプログラムまで，同じ代数的枠組みで記述でき，仕様の段階的な実現（具体化）が可能である．

代数的仕様記述法における検証は，一般には，仕様に要求される所期の性質が，その公理系の定理として成立することを示すことである．特に仕様を段階的に実現する場合は，抽象度の高い抽象データ型の演算が抽象度の一段低いデータ型の演算を用いて正しく実現されていること，すなわち下位の仕様が，上位の仕様の公理を満たしていることを示すことである．

通常，代数的仕様の検証では，与えられた仕様の公理系を項書き換え系とみなして，項を書き換えることにより，検証作業を進めていくが，その検証作業は煩雑である．そのため，項の書き換え等の作業を実行することを目的とする検証支援系が幾つか提案されているが，実際的規模の検証に対しては，機能が十分でないのが現状である．本論文の前半では，項書き換え系とみなせるクラスの代数的仕様を主として対象とする代数的仕様検証支援系の概要について述

べている．本支援系は，従来の支援系に比べ，プレスブルガーの算術に関する性質を用いて検証を行う場合，検証が機械的に行なえるという特徴をもつ．

又，従来，代数的に書かれた仕様の検証例として引き合いに出されるものは，配列と整数の対で正しくスタックが実現されるか等，非常に簡単なものが多く，実用的な規模の問題に対する検証例がほとんど報告されていない．そこで本論文の後半では，代数的に記述されたハイレベルデータリンク制御（HDLC）手順の検証結果について報告する．

第１章では，文献[1]，[2]として公表した代数的仕様検証支援系の概要及び支援系を用いた検証例について述べられている．筆者らの支援系は，AFFIRMシステム[2]等と同様，主として項書き換え系上での検証を支援するシステムであるが，(1) 与えられた項書き換え系の Church-Rosser性や有限停止性を前提としない，(2) プレスブルガー文の真偽判定機能を持ち，ブール代数や加減算，大小比較を持つ整数の性質を用いて検証を行う場合，検証が機械的に行える，(3) 記述言語の変更に容易に対応できる，等の特徴を持つ．

従来，項書き換え系上で検証を行う場合，検証が形式的に行える反面，ブール代数や整数の性質など基礎的な数学の諸性質についても，すべて検証者が公理として導入しなければならず，検証が煩雑になる等の問題点があった．筆者らの支援系では，プレスブルガーの算術に関する性質（ブール代数や整数の大小比較に関する幾つかの基本的性質を含む）のみを用いて検証できる部分は，検証が完全に機械的に行えるよう，プレスブルガー文の真偽判定機能を組み込んでいる．第１章の後半で述べるクイックソートプログラムの検証や第２章で述べるHDLC手順の検証，Stenningのプロトコルの検証[8]等では，ほとんどの公理がプレスブルガー算術の演算を用いて記述されているため，この機能を有効に利用できた．

第２章では，文献[3]～[6]として公表したハイレベルデータリンク制御（HDLC）手順の代数的記述とその検証結果について述べられ

ている．一般に，HDLC手順のような伝送制御手順（プロトコル）の記述法は，状態遷移図を利用する方法と，プログラム言語を利用する方法とに大別される．前者の記述法を用いた時の検証における問題点は，プロトコルの複雑化に伴い，状態数が急激に増加することであり，後者の問題点は，処理順序の一意的記述により過剰規定に陥り易いことや，利用する言語の意味の形式的定義が複雑であることである．代数的仕様記述法は，状態遷移図等では取り扱いの困難なシーケンス番号やパラメータ値の取り扱いが簡単に記述でき，又，一般のプログラム言語に比べ意味の形式的定義が容易であるなど，上述の欠点を免れ得る有効な手法の１つであると考えられる．

筆者らは，まずHDLC手順の規格を分析し，その結果に基づきHDLC手順を用いて通信を行う通信系を抽象的な順序機械とみなして，その代数的記述を作成した．すなわち，通信系に対し，内部構造を陽に持たない抽象的な”状態”を導入し，送信や受信といった動作を”状態遷移関数”として，又，各局で保持すべき変数（状態変数等）の値を状態からの”出力関数”として導入した．そして，それぞれの状態遷移後における各出力関数の値が，遷移前の状態における各出力関数の値からどのように定まるかを公理として表した．次に，この記述が公理系として”矛盾なく”且つ”不足なく”書かれていることを示すと共に，この記述において成立する幾つかの重要な論理関係式（例えば，通信系全体の動作可能性や１，２次局の状態変数間に成立する関係等）の検証を行った．検証は主として構造的帰納法を用いて行い，検証作業には第１章で述べた検証支援系を利用した．

関連発表論文

[1] 東野，工藤，縄田，杉山，谷口：”代数的仕様検証支援系及びそれを用いた検証例”，電子通信学会論文誌(D)（昭和59年4月掲載予定）．

[2] 東野，工藤，杉山，縄田：”代数的仕様の検証支援システムの試作”，電子通信学会技術研究報告，AL82-27（昭和57年9月）．

[3] 東野，森，谷口，嵩：”代数的に記述されたHDLCプロトコルの検証”，電子通信学会論文誌(D)，J66-7，7，pp.773-780（昭和58年7月）．

[4] 東野，森，杉山，谷口，嵩：”HDLCの代数的記述と検証”，電子通信学会技術研究報告，AL80-07（昭和55年5月）．

[5] 東野，縄田，森，谷口，嵩：”代数的記述における検証について－HDLC手順の場合－”，電子通信学会技術研究報告，AL81-71（昭和56年11月）．

[6] T.Higashino, M.Mori, Y.Sugiyama, K.Taniguchi, T.Kasami: ”An Algebraic Specification of HDLC Procedures and its Verification”, IEEE Transactions on Software Engineering (to appear).

第１章　代数的仕様検証支援系

１．１　序言

代数的記述言語は意味の定義が簡潔であり，検証が形式的に行える等の利点がある．特に与えられた公理系が項書き換え系[1]とみなせるクラスに対しては，幾つかの検証法[1]や検証支援系[2][3]が提案されている．

この章では，筆者らが定義した代数的記述言語ASL/V[4] による仕様の作成，及び作成した仕様上での検証の各過程を対話的に支援することを主な目的とする代数的仕様検証支援系の概要及び支援系を用いた検証例について述べる．

筆者らの支援系は，AFFIRMシステム[2]や川原林らのシステム[3]同様，主として項書き換え系上での検証を支援するシステムであるが，(1) 与えられた項書き換え系の Church-Rosser性や有限停止性を前提としない，(2) プレスブルガー文[5]の真偽判定機能を持ち，ブール代数や加減算，大小比較を持つ整数の性質を用いて検証を行う場合，機械的に検証が行える，(3) 記述言語の変更に容易に対応できる，等の特徴を持つ．

一般に項の書き換えによって検証を行う場合，ブール代数や整数などに関する基礎的な数学の諸性質についても，すべて検証者が補助定理として導入しなければならず，検証が煩雑になり，また，交換則や結合則を導入すると，与えられた項書き換え系の有限停止性が失われてしまう等の問題点があった．筆者らの支援系では，プレスブルガーの算術に関する性質（ブール代数や整数の大小比較に関する幾つかの基本的性質を含む）を用いて検証を行う場合，検証が機械的に行え，上述の問題点が改善される．例えば，１．４で述べるクイックソートプログラムの検証やHDLC手順の検証[6]，Stenningのプロトコル[7]の検証[8]等の場合，項の書き換えのみによる方法に比べ検証に要した全作業時間を短縮できた（１．５参照）．

1．2　諸定義

1．2．1　代数的記述

以下，本論文の基本的定義は文献(9)に従う．仕様Dは，3字組（S，Σ，E）で定義する．S，Σ，Eはそれぞれソート，関数記号，公理の集合である．公理は同一ソートの（一般に変数を含む）項の順序対で，"l=r"で表す．公理の集合を公理系と呼び，公理系Eによって生成される合同関係[8]を"$\equiv_E$"で表す．$t \equiv_E t'$ならば，tとt′は等価であると言う．2つの仕様D1=(S1, Σ1, E1), D2=(S2, Σ2, E2)に対し，S1⊇S2，Σ1⊇Σ2，E1⊇E2の時，D2をD1の部分仕様と呼ぶ．

変数記号の集合をV，変数を含む項の集合をT(Σ+V)，変数を含まない項の集合をT(Σ)で表す．Σを構成子の集合Σ^cと定義関数の集合Σ^dに分割する．構成子だけから成る項を構成子項と呼ぶ．以下では，ソートsの構成子項の集合$T(\Sigma^c, s)$がソートsの値の集合を表すと考える．このため，相異なる2つの構成子項t，t′に対して$t \equiv_E t'$が成立しない時，公理系Eは無矛盾であると呼ぶ[8]．又，$f \in \Sigma^d$, $t_i \in T(\Sigma^c, s_i)$ $(1 \leq i \leq n)$に対して，$t_0 \equiv_E f(t_1, t_2, \ldots, t_n)$であるような構成子項$t_0$が唯一つ存在する時，引数$t_1, t_2, \ldots, t_n$に対するfの値が$t_0$であると呼び，このような$t_0$が存在しない時，fの値は定義されないと言う．各引数ソートの任意の構成子項$t_1, t_2, \ldots, t_n$に対して，f（あるいは，一般に項t）の値が定義される時，f（又は項t）は構成子項全域で定義されていると言う．

同一ソートの項（一般に変数を含む）の対l，rに現れる各変数への任意の構成子項の代入[8] σに対して，$\sigma(l) \equiv_E \sigma(r)$が成立する時†，"$l \cong_E r$"と書き，$l \cong_E r$を公理系Eの"定理"と呼ぶ．もちろん，$l \equiv_E r$ならば$l \cong_E r$であるが，逆は一般に成り立たない．

---

†　"σ(l)，σ(r)の値が定義されている時，$\sigma(l) \equiv_E \sigma(r)$"のような条件を採用してもよいが，ここでは本文中の定義を採用する．

仕様を書く上での基本演算は，ブール代数や加減算を持つ整数などといった具体的な代数[9]における関数として定義されているとし，このような代数を基底代数と呼ぶ．基底代数の仕様DB=(SB.ΣB.EB)においては，各公理が”f(d1,d2,....,dn)==d0”（但し，各diはそのソートの値（構成子項），fはΣBの定義関数）の形をし，各関数の関数表が公理の形で与えられていると概念上考える（一般に公理は無限個）．

1．2．2　項書き換え系

項の書き換え規則は，同一ソートの項l，rの順序対であり，”l→r”で表す．この時，項rに現れる変数はすべて項lにも現われていなければならない．又，書き換え規則の集合を項書き換え系と呼ぶ．項書き換え系Rの書き換え規則”l→r”と項t1の部分項t1′，及びt1′=σ(l)を満たす代入σが存在し，t2がt1の部分項の出現[9]t1′をσ(r)で置き換えて得られる項である時，且つその時のみ，”$t1 \underset{R}{\Rightarrow} t2$”と書き，項t1は項書き換え系Rの書き換え規則を用いて項t2に書き換えられると言う．”$\overset{*}{\underset{R}{\Rightarrow}}$”を関係”$\underset{R}{\Rightarrow}$”の推移反射閉包とする．$t \overset{*}{\underset{R}{\Rightarrow}} t'$を満たす項t，t′に対して，$t' \underset{R}{\Rightarrow} t''$を満たす項t″が存在しない時，t′をtの標準項といい，tの1つの標準項を”t↓”で表す．任意の項t0に対して，$t0 \underset{R}{\Rightarrow} t1 \underset{R}{\Rightarrow} t2 \underset{R}{\Rightarrow} \cdots$であるような無限系列が存在しない時，Rは有限停止性を持つと言う．

公理系Eの各公理の右辺に現れる変数が左辺にも現れる時，公理系Eの各公理を左辺から右辺への書き換え規則として扱うことにより，Eを項書き換え系とみなすことが出来る．公理系Eを項書き換え系として扱うとき，これをR_Eで表す（すなわち，$R_E=\{\alpha\rightarrow\beta \mid \alpha==\beta\in E\}$）．$t \underset{E}{\equiv} t'$なる任意の項t，t′に対して，$t \overset{*}{\underset{R_E}{\Rightarrow}} t''$且つ$t' \overset{*}{\underset{R_E}{\Rightarrow}} t''$を満たす項t″が存在する時，$R_E$はChurch-Rosserであるという．

2個の書き換え規則l1→r1，l2→r2に対して，l1′を変数のみで

ない１つのl1の部分項とする．l1，l2に対する項の代入σのうちで σ(l1′)＝σ(l2) を満たす代入σが存在する時，このような代入のうち最も一般的な代入[1] σ′ に対して，σ′(l1) の部分項σ′(l1′) をσ′(r2) で書き換えた項とσ′(r1) の対をcritical pair と呼ぶ．定義より，R_E上の任意のcritical pair <p, q> 及び $p \overset{\text{数}}{\underset{R_E}{\Rightarrow}} p'$，$q \overset{\text{数}}{\underset{R_E}{\Rightarrow}} q'$ を満たす任意の項p′，q′について，$p' \overset{\text{言}}{\underset{E}{=}} q'$が成立する．

１．２．３　代数的記述言語ASL/Vの概要

筆者らが定義した代数的記述言語ASL/V の概要について，表２†のクイックソートプログラムの記述を例に説明する．" SPEC QUICK SORT. LEVEL4 " は以下が仕様名" QUICKSORT. LEVEL4" の仕様記述であることを表す．" INCLUDE PRESBURGER, ARRAY" は，別に分離して記述された仕様PRESBURGER 及びARRAY（表１††）を部分仕様として取り込む事を指定した文であり，" OP QS:array, integer, integer ->array " は，定義関数QSの第1, 2, 3 引数のソートがarray であることの宣言である（尚，構成子の宣言は，" CON ..." を用いる）．" VAR x SORT array" は，変数名ｘの属するソートがarray であることを表す．公理は" ラベル: t==t′ " で表す．尚，ソート名の宣言は，表1(1)のPRESBURGERの仕様のように，" SORT bool, integer" のように行う．

尚，代数的記述言語ASL/V の詳細については，文献(4)参照．

---

†　20ページ参照．

††　19ページ参照．

1．3　支援系の構成

以下では，定理の証明を検証の対象とする．本支援系では，代数的記述言語ASL/V で記述された仕様D=(S, Σ, E)上で，" $t \underset{E}{\approx} t'$ " の証明の各過程を支援することを主な目的としている．

1．3．1　支援系の概要

支援系は，代数的記述言語ASL/V で書かれた仕様の入力，編集，構文解析等を行う仕様作成部と定理の証明の各過程を支援する検証支援部に大別される．現在，その支援系は，大阪大学大型計算機センターのACOS1000のTSS 環境下で稼働している．仕様作成部は言語XXPL[10]で記述され，検証支援部は言語PL/Iで記述されている．ソースプログラムのサイズは約5000行，使用メモリーサイズは約150 キロ語であり，作成に要した労力は約14人月である．

検証支援部では，利用者は次の(a)から(f)のような機能を用いて対話的に検証を行う．

(a) 項書き換え機能

(b) Knuth-Bendixの完全化アルゴリズム

(c) 場合分けの機能

(d) プレスブルガー文真偽判定機能

(e) 帰納法による証明支援機能

(f) 公理系の矛盾検出機能

これらの支援機能の詳細及び幾つかの証明法については，次節以降で説明する．

1．3．2　公理系が項書き換え系とみなせる場合の証明

与えられた公理系Eが項書き換え系とみなせる場合，2項 t，t′ に対し $t \overset{*}{\underset{R_E}{\Rightarrow}} t''$，$t' \overset{*}{\underset{R_E}{\Rightarrow}} t''$ を満たす項t″が存在すれば，$t \underset{E}{\equiv} t'$ である．特に，R_Eの有限停止性及びChurch-Rosser が保証されれば，$t \underset{E}{\equiv} t'$ と $t\downarrow = t'\downarrow$ は等価である[1]．そこで，支援系では次のような機能を提供する．

(1) 項書き換え機能

項 t 及び書き換え規則（公理）及び書き換え規則を適用する位置（項 t の出現[9]）を指定すると，指定に従った書き換えを実行する．又，書き換え回数の上限 n を指定すると，（n 回以下で）可能な限り書き換えを実行し，結果を表示する（書き換えの順番は，項の外側からの適用を優先する）．

(2) 有限停止性に関する機能

一般に与えられた項書き換え系が有限停止性を持つかどうかの判定問題は決定不能[1]であるが，有限停止性を保証するための幾つかの十分条件が知られている．本支援系では，その十分条件の１つである一般帰納的経路順序法[1]を採用し，与えられた項書き換え系がこの十分条件を満足するか否かを判定する．

(3) critical pair 表示機能

一般に項書き換え系R_Eが有限停止性を持つ場合，R_Eで生成されるすべてのcritical pair <pi, qi> に対し，pi↓=qi↓ ならば，R_Eの Church-Rosser 性が保証される．支援系では，R_E上で生成されるすべてのcritical pair <pi, qi> あるいは，それらの１つの標準項の対<pi↓, qi↓> を表示する機能を持つ．

(4) Knuth-Bendixの完全化アルゴリズム

Church-Rosser 性の保証されていない項書き換え系R_Eを合同関係を変えずに，有限停止性，Church-Rosser 性の保証された項書き換え系R_{E1}に変換するための有効なアルゴリズムとして，Knuth-Bendix の完全化アルゴリズム[1]が知られている．支援系には，そのアルゴリズムが組み込まれているが，このアルゴリズムはcritical pair が無限に発生して停止しない場合やアルゴリズムが失敗に終る場合がある[1]．このため，本支援系では，アルゴリズムを途中で中断し，その時点で生成された項書き換え系R_{E2}上でt↓=t′↓か否かを判定（R_{E2}上でt↓=t′↓ならば，$t \equiv_E t'$ である[4]）できるようにしたり，発生したcritical pair から検証者が不必要と思うcritical pair を消去できるようにする等，検証者が介入できるよう設計した（詳

細は，文献(4)参照）．

(5) 定理(補題)の追加機能

支援系では証明済みの定理 $t \cong_E t'$ を公理 $(t==t')$ として追加できる．追加した定理を書き換え規則として適用する場合は，定理の各変数に代入される項の値が構成子項全域で定義されていなければならない．

1．3．3　場合分けによる証明

簡単のため，$p(x) \cong_E TRUE$（但し，$p(x)$の値域ソートはブールとし，公理系Eは無矛盾とする．又，xは一般には幾つかの変数を表すとする）の証明において，$p(x)$が部分項として$q(x)$（但し，$q(x)$の値域ソートはブールとする）を含む場合（例えば，1．4．1で述べるif関数を用いて，$p(x)=IF\ q(x)\ THEN\ p1(x)\ ELSE\ p2(x)$ と表せる場合）について説明する．以下，$q(x)$は構成子項全域で定義され，且つ $NOT(q(x) \cong_E TRUE)$，$NOT(q(x) \cong_E FALSE)$ とする．この時，変数xと同一ソートの新たな定数関数"X"（xのソートの任意の構成子項を表すと考える）を導入し，$E1=E\ U\ \{q(X)==TRUE\}$，$E2=E\ U\ \{q(X)==FALSE\}$ の2つの公理系に対し，(イ) $p(X) \cong_{E1} TRUE$，(ロ) $p(X) \cong_{E2} TRUE$ が共に成立すれば$p(x) \cong_E TRUE$である．尚，xの値によらず$q(x) \cong_E TRUE$（又は，FALSE）の場合，条件(イ)（又は，条件(ロ)）のみが成立すればよい．

支援系では，証明すべき項p及び場合分けしたい部分項qを指定すると，公理系Eに"q==TRUE" 又は，"q==FALSE"を追加した2つの公理系E1，E2を順次作成し，R_{E1}，R_{E2}上で項pの1つの標準項を求め，これを表示する．場合分けの項は最大20個指定出来，すべての場合分けの組合せに対し，結果を表示する．

1．3．4　プレスブルガーの算術を利用した証明

以下，公理系Eがプレスブルガーの算術に関する公理を（基底代数として）含む場合について考える．プレスブルガーの算術とは，

整数，整数上の変数，加減算（＋，－），大小比較（<, ≦, >, ≧, ＝），ブールの論理関数（AND, OR, NOT）及び限定作用素（∀，∃）から構成される算術であり，この算術のクラスに含まれる論理式のうち，自由変数を含まないものをプレスブルガー文と呼ぶ．本論文では，プレスブルガー文Ｐの定義を拡張し，Ｐがブール値をとる変数を含んでもよいとする．

さて，項ｔに対し，t≒TRUE であることをプレスブルガー文の真偽判定機能を用いて証明する方法について述べる．たとえば，
t={(f(y)≧g(0)) OR (f(y)<g(0)+2) OR h(z)} の場合，f(y), g(0), h(z)のようにプレスブルガーの算術に含まれない関数記号を最も外側に持つ部分項をそれぞれ変数x1, x2, w1に置き換えると，項ｔは，{(x1≧x2) OR (x1<x2+2) OR w1}（以下，Pt(x1, x2, w1) で表す）のような項に変換できる．逆に，項Pt(x1, x2, w1)を用いて項ｔは，Pt(f(y), g(0), h(z))と表せる．この時，次の(i)，(ii) が成立すれば，t≒TRUE が成立する（証明は定義より明らか）．

(i) プレスブルガー文∀x1x2w1 Pt(x1, x2, w1) が”真”である．

(ii)部分項f(y), g(0), h(z)の値が構成子項全域で定義されている．

一般にプレスブルガー文の真偽判定問題は決定可能[5]であり，支援系では証明すべき定理t≒TRUE に対し，項ｔから上述のような項Pt(x1, x2, ..., xk)を自動的に作成し，次にこのプレスブルガー文∀x1x2...xk Pt(x1, x2, ..., xk)の真偽を判定している．以下，この真偽判定について説明する（但し，説明を簡単にするため，以下では項Ptにはブール値をとる変数は含まないとする）．

∀x1x2...xk Ptが”真”であることを示すには，NOT(Pt) を満たす整数解が存在しないことを示せばよい．そこで，NOT(Pt) を次のような積和形に展開する．

NOT(Pt) = Q1 OR Q2 OR ... OR Qn

Qi = Ri1 AND Ri2 AND ... AND Rimi

Rij = tij ◇ tij′

（但し，”◇”は，≧，>，≦，<，＝のいずれかの関数記号．又，

tij, tij′ は，整数，整数値をとる変数，及び＋，－から構成される項）．

NOT(Pt) が整数解を持たないことと，すべてのQiが整数解を持たないこととは等価であるから，各Qiが整数解を持たないかどうかを判定すればよい．整数解非存在のテストは一般に時間がかかり，又，整数計画問題を解くルーチンが手許になかったこともあって，現在，本支援系では，テスト的に実数に対するシンプレックス法を利用している．そして，各Qiがシンプレックス法で実数解を持たないと判断された時のみ∀x1x2...xk Ptを”真”と判定している（実数解を持つ時は，判定不明とする）．この方法は，整数解を持たないための十分条件の判定であるが，これまでに筆者らが検証した[6][8]定理の多くが実数上でも成立するタイプの定理であり，更に上述の判定不明が生じにくいように，”x<y”の形の項を”x≦y-1”に変形する等の処置を講じたので，筆者らの検証例ではこの十分条件で証明が成功した．又，シンプレックス法による実数解非存在の判定は，比較的短時間で実行できる．現在，上述の判定不明の場合に対し，整数解の存在を判定する機能の導入を検討している．

尚，計算機を用いて数値計算を行う際の誤差から生じる誤判定を防ぐため，シンプレックス法実行時のイタレーション回数を制限する等の処置を講じている[13]．

１．３．５　構造的帰納法による証明

簡単のため，p(x)≅TRUEの証明を変数x（ソートをsとする）についての構造的帰納法で行う場合について説明する．ソートsの定数関数（構成子）をci（1≦i≦l），定数関数以外でソートsを値域とする構成子をgj（1≦j≦m）で表す．説明を簡単にするため，各gjはソートsの引数をちょうど1個持ち，それ以外の引数ソートは持たないとする．この時，次の(i)，(ii)が成立すれば p(x)≅TRUEである．

(i) 初期段階：各定数関数ciに対し，p(ci)≡TRUE である．

(ii)帰納段階：定数関数以外のすべての構成子gjに対し，(p(x)⊃p(gj(x)))≅$_E$TRUEである（"⊃"は，ブールの"含意"を表し，表1(3)の公理を持つ）．

通常，(p(x)⊃p(gj(x)))≅$_E$TRUEの証明は，1．3．3で述べた場合分けの方法を用いて行う．すなわち，項の深さがn段であるようなソートsの任意の構成子項を表す定数関数"X"を導入し，帰納法の仮定として，公理系Eにp(X)==TRUEなる公理を追加する（追加した公理系をE1とする）．次に公理系E1上で，各構成子gjに対して，p(gj(X))≅$_E$TRUEを証明する．

支援系では，初期段階の証明に対して，項p(x)と定数関数c1, c2, ...cnを与えると，項p(c1), p(c2), ...p(cn)を順次生成し，帰納段階の証明に対して，項p(x)と関数g1, g2, ...gmを与えると，変数xの属するソートに新たな定数関数"X"を導入し，項p(g1(X)), p(g2(X)), ...p(gm(X))を順次生成する機能を持つ．

1．3．6　関数fが帰納的に定義されている場合のQ(x, f(x))≅$_E$TRUEの形の定理の証明

f(x)≅$_E$IF p(x) THEN k(x) ELSE f(g(x))であり，且つ，次の(1)から(3)の条件が成立するならば，Q(x, f(x))≅$_E$TRUEである．

(1) Q(x, f(x))に対応して検証者が選んだ適当な述語R(x, y)（但し，x，yは関数fの引数ソートと同一ソートの変数）に対し，以下の3条件が成立する．

(i)　R(x, x)≅$_E$TRUE

(ii)　{(R(x, y) AND p(y))⊃Q(x, k(y))}≅$_E$TRUE

(iii)　{(R(x, y) AND NOT(p(y)))⊃R(x, g(y))}≅$_E$TRUE

(2) 変数xと同一ソートの任意の構成子項Xに対して，ある非負整数nが存在して，p(g^n(X))≡$_E$TRUE，且つ0≦i<nを満たす任意のiについて，p(g^i(X))≡$_E$FALSEである（但し，g^j(X)=g(g^{j-1}(X))，g^0(X)=Xとする）．

(3) p(Y)≡$_E$FALSEを満たす任意の構成子項Yに対し，g(Y)の値が定義されている．

尚，この方法は，１．４で述べるクイックソートプログラムの検証に利用する．

１．３．７　公理系の矛盾検出機能

１．２の定義より，相異なる２つの構成子項ｔ，ｔ′が等価になる公理系は矛盾である．支援系では，t==t′ なる公理を利用者が指定したり，１．３．２で述べたKnuth-Bendixの完全化アルゴリズムの実行中ｔ≡ｔ′なる関係が発見された時，公理系の矛盾を利用者に知らせる．

尚，１．３．２から１．３．７で述べた機能以外の検証支援機能（例えば，公理の属性指定機能，critical pair を両辺とする公理の追加機能など）については，文献(4)参照．

１．３．８　仕様作成部の概要

仕様作成部では，主に次のような機能を提供する（コマンドの詳細については，文献(4)参照）．

(1) 仕様の入力，編集のための機能

(2) 作成した仕様をファイルへ格納する機能

(3) 作成した仕様の構文解析機能

(4) 作成した仕様を検証支援部で扱いやすいような内部表現へ変換する機能

尚，仕様作成部では以下の点が考慮されている．

(A) 代数的記述言語ASL/V の構文規則の変更が生じても，構文解析プログラムの修正が容易に行えるよう言語XXPL[10]で記述した．この言語には，BNF 記法で書かれた文法に対して，構文解析に必要なテーブル類を出力する文法アナライザーが用意されており，ASL/V の変更に比較的容易に対応できる．

(B) INCLUDE 文で包含される部分仕様としては，パラメータ付のものも許す．又，部分仕様ごとに，入力，構文解析，ファイルへの格納が出来る．尚，扱えるソート名，関数名，公理の最大個数は，それぞれ50個，250 個，2000個である．

１．４　クイックソート プログラムの検証

以下では，ASL/V で記述したクイックソートの関数型プログラム（表２）がソーティングプログラムに要求される諸性質を満たしていることを，検証支援系を用いて証明した概略について述べる．

１．４．１　基底代数について

以下では，次の(1) から(3) を基底代数として用いる（表１）．

(1) プレスブルガーの算術：ブール値及び整数値を表すソートをbool, integerと書く．ソートboolの値はTRUE, FALSEでソートintegerの値は0, 1, -1, 2, -2, ... で表す．又，AND, OR, NOT, =, ≧, >, ≦, <の関数を用いる．尚，これらに関する演算は通常の定義に従う．

(2) 整数を要素とする配列：配列の集合を表すソートをarray と書く†．配列ｘのｉ番目の要素をCONTENT(x, i)で，配列ｘのｉ番目とｊ番目の要素を交換して得られる配列をEXCH(x, i, j) で表し，配列ｘのｉ番目からｊ番目の要素のうちで，整数ｖと等しい要素の個数をNUM(x, i, j, v)で表す．配列ｘ及び整数ｉ，ｊの３字組を<x, i, j>で表し，そのソートをarrintint で表す．３字組<x, i, j> の各要素ｘ，ｉ，ｊをそれぞれPRk(<x, i, j>) (但し k=1, 2, 3) で表す．

(3) 上述の(1)，(2)において，表１(3) の公理を持つif関数が暗黙のうちに宣言されているとする．但し，if.bool ，if.integer ，if.array, if.arrintintをif.sでまとめて表す．尚，以下では if.bool(x, y, TRUE) を” x⊃y ” で表す．又，混同がない限り if.s(t1, t2, t3)を” IF t1 THEN t2 ELSE t3 ” で表す．

１．４．２　クイックソートの関数型プログラム

表２（レベル４）は，クイックソートの関数型プログラムの例で

---

†　ソートarray は整数を要素とする配列からなる集合と考えている．従って，各配列がソートarray の構成子項となる．

表1　基 底 代 数

```
(1) SPEC   PRESBURGER ;
     SORT bool, integer ;
     CON  TRUE, FALSE          : -> bool
          0, 1, -1, 2, -2, ...  : -> integer ;
     OP   AND, OR : bool, bool -> bool,
          NOT     : bool       -> bool,
          +, -          : integer, integer -> integer,
          =, >, ≧, <, ≦ : integer, integer -> bool ;
   END ;

(2) SPEC   ARRAY ;
     INCLUDE PRESBURGER ;
     SORT array, arrintint ;
     OP   CONTENT : array, integer          -> integer,
          EXCH    : array, integer, integer -> array,
          NUM     : array, integer, integer, integer
                                            -> integer,
          < , , > : array, integer, integer -> arrintint,
          PR1     : arrintint -> array,
          PR2, PR3: arrintint -> integer ;
     VAR  x    SORT array,
          i, j SORT integer ;

     AR1: PR1(<x, i, j>) == x ;
     AR2: PR2(<x, i, j>) == i ;
     AR3: PR3(<x, i, j>) == j ;
   END ;

(3) if-function
     AX1: if.s(TRUE, x, y) == x ;
     AX2: if.s(FALSE, x, y) == y ;
     AX3: x ⊃ y == if.bool(x, y, TRUE) ;
```

図1　SPRIT の説明図

	l	l+1	・・・		h-1	h
array　x	25	10	33	47	21	38
PR1(SPRIT(...))	21	10	33	47	25	38
		↑ P2	↑ P3			

表2　クイックソート（レベル4）

```
SPEC    QUICKSORT.LEBEL4 :
  INCLUDE  PRESBURGER, ARRAY ;
  OP     QS     : array, integer, integer -> array,
         SPRIT : arrintint, integer -> arrintint,
         LEFT, RIGHT : array, integer, integer -> integer ;
  VAR    x      SORT array,
         l, h, b SORT integer ;
  AX1:   QS(x, l, h) == IF h ≦ l THEN x
           ELSE QS(QS(PR1(SPRIT(<x, l, h>, CONTENT(x, l))),
                    l, PR2(SPRIT(<x, l, h>, CONTENT(x, l)))),
                PR3(SPRIT(<x, l, h>, CONTENT(x, l))), h)  ;
  AX2:   SPRIT(<x, l, h>, b) == IF h ≦ l THEN <x, h, l>
           ELSE IF RIGHT(x, h, b) < LEFT(x, l, b)
                THEN <x, RIGHT(x, h, b), LEFT(x, l, b)>
           ELSE SPRIT(<EXCH(x, RIGHT(x, h, b), LEFT(x, l, b)),
                  LEFT(x, l, b)+1, RIGHT(x, h, b)-1>, b)  ;
  AX3:   LEFT(x, l, b) == IF CONTENT(x, l) ≧ b THEN l
           ELSE LEFT(x, l+1, b)  ;
  AX4:   RIGHT(x, h, b) == IF CONTENT(x, h) ≦ b THEN h
           ELSE RIGHT(x, h-1, b)  ;
END ;
```

表3　クイックソート（レベル1）

```
SPEC  QUICKSORT.LEBEL1 :
  INCLUDE PRESBURGER, ARRAY ;
  OP     QS: array, integer, integer -> array ;
  VAR    x          SORT array,
         l, h, i, j, v SORT integer ;
  SP1:   [ l ≦ i AND i ≦ j AND j ≦ h ] ⊃
           CONTENT(QS(x, l, h), i) ≦ CONTENT(QS(x, l, h), j) == TRUE ;
  SP2:   NUM(QS(x, l, h), l, h, v) = NUM(x, l, h, v) == TRUE ;
  SP3:   [ i < l OR h < i ] ⊃
           CONTENT(QS(x, l, h), i) = CONTENT(x, i) == TRUE ;
END ;
```

あり，配列X，及び整数値L，Hに対し，表2の公理系を項書き換え系とみなし，QS(X,L,H)の標準項を求めると，配列XのL番目からH番目まで（以下［L，H］で表す）の要素をソーティングした配列が求まる．又，表2のような関数型プログラムに対して最適化コンパイラー[11]が試作されており，表2の例では，Wirth によるPascalで書かれたプログラム[12]とほぼ同程度の効率の良いオブジェクトプログラムを生成する[11]．

表2では，関数QS以外にSPRIT,LEFT,RIGHTを用いる．関数SPRITの値は3成分より成り，例えば図1の配列x及び境界値25に対し，第1成分 PR1(SPRIT(<x,l,h>,25)) は，図1の下側のような配列を表す．この配列は境界値25以下の要素から成る左部分［l，P2］と，境界値25以上の要素から成る右部分［P3，h］に分割されており，第2第3成分PR2(SPRIT(<x,l,h>,25))，PR3(SPRIT(<x,l,h>,25))がそれぞれ上述のP2，P3を表す．公理AX1は，［l，h］間のクイックソートを，分割後の配列の2つの部分［l，P2］間及び［P3，h］間の2回のクイックソートとして実行することを表している．尚，本論文では境界値として，配列xの1番目の要素CONTENT(x,1)を用いている．

関数LEFT(x,l,b) は，配列xの1番目より右側† で，境界値b以上の要素がはじめて現れる場所を表し，RIGHT(x,h,b)は，配列xのh番目より左側† でb以下の要素がはじめて現れる場所を表す．2つの関数は，SPRIT の値を帰納的に定義するために用いられている．

1．4．3　ソーティングプログラムが満たすべき性質

表3（レベル1）は，最も抽象度の高いソーティングの仕様例であり，3個の公理はそれぞれ(SP1) ソーティング後の配列の要素は昇順に並んでいる，(SP2) ソーティングの前後で配列に含まれる要

---

†　配列の要素は図1のように左から右へ順に並んでいると考える．

素は多重集合† として変らない，(SP3) ソーティングの範囲外の配列の要素は不変である，を記述したものである．

以下では，表２のクイックソートプログラムが上述の３個の性質を満たしていること，すなわち，表２の仕様が表３の仕様の実現であることを証明している．尚，仕様の実現については，次の定義に従う．

［定義１］公理系E2，無矛盾な公理系E1に対し，(a) 公理系E2が無矛盾で，且つ (b) E1の各公理 ti==ti′ に対し，ti≅ti′ が成立する時，E2はE1の実現であるという．

１．４．４　検証の具体的方法

１．４．４．１　中間レベルの仕様

表２（レベル４）の仕様が表３（レベル１）の仕様の実現であることを直接証明するのは煩雑であるため，表４（レベル２），表５（レベル３）のような２つの中間的な仕様を記述し，それぞれレベル i+1 がレベル i (i=1, 2, 3) の実現であることを示すことにより，レベル４がレベル１の実現であることを証明する[13]．

表４は，レベル２がレベル１の実現であることを証明するのに必要と思われる関数SPRIT に関する７個の性質を記述した．次に各公理を簡単に説明する．(TH1) 分割幅が２以上の時，P3†† の値は１より大きい，(TH2) P2†† の値は h より小さい，(TH3) P2の値はP3の値より小さい，(TH4) 分割の範囲外の配列の要素は不変である，(TH5) 分割の前後で配列の要素は多重集合として変らない，(TH6) 分割後の配列の右部分の要素は，境界値CONTENT(x, l)以上である，(TH7) 分割後の配列の左部分の要素は境界値以下である．

---

†　集合の中に同一要素が２個以上現れることも許す．

††　P2はPR2(SPRIT(<x, l, h>, CONTENT(x, l)))を
　　P3はPR3(SPRIT(<x, l, h>, CONTENT(x, l)))を表すとする．

表4　クイックソート（レベル2）

```
SPEC    QUICKSORT.LEBEL2 ;
  INCLUDE  PRESBURGER, ARRAY ;
  OP    QS     : array, integer, integer -> array,
        SPRIT  : arrintint, integer -> arrintint ;
  VAR   x        SORT array,
        l, h, i, v SORT integer ;
  AX1:  QS(x, l, h) == IF h ≦ l THEN x
          ELSE QS(QS(PR1(SPRIT(<x, l, h>, CONTENT(x, l))),
                  l, PR2(SPRIT(<x, l, h>, CONTENT(x, l)))),
               PR3(SPRIT(<x, l, h>, CONTENT(x, l))), h) ;
  TH1:  l < h ⊃ l < PR3(SPRIT(<x, l, h>, CONTENT(x, l))) == TRUE ;
  TH2:  l < h ⊃ PR2(SPRIT(<x, l, h>, CONTENT(x, l))) < h == TRUE ;
  Th3:  l < h ⊃ PR2(SPRIT(<x, l, h>, CONTENT(x, l))) <
                    PR3(SPRIT(<x, l, h>, CONTENT(x, l))) == TRUE ;
  TH4:  [ l ≦ h AND (i < l OR h < i) ] ⊃ CONTENT(x, l) =
          CONTENT(PR1(SPRIT(<x, l, h>, CONTENT(x, l))), i) == TRUE ;
  TH5:  l ≦ h ⊃ NUM(x, l, h, v) =
          NUM(PR1(SPRIT(<x, l, h>, CONTENT(x, l))), l, h, v) == TRUE ;
  TH6:  [ l ≦ AND PR2(SPRIT(<x, l, h>, CONTENT(x, l))) < i
          AND i ≦ h ] ⊃ CONTENT(x, l) ≦
            CONTENT(PR1(SPRIT(<x, l, h>, CONTENT(x, l))), i) == TRUE ;
  TH7:  [ l ≦ h AND l ≦ i AND
          i < PR3(SPRIT(<x, l, h>, CONTENT(x, l))) ] ⊃
          CONTENT(PR1(SPRIT(<x, l, h>, CONTENT(x, l))), i)
            ≦ CONTENT(x, l) == TRUE ;
END ;
```

表5　クイックソート（レベル3）

```
SPEC    QUICKSORT.LEBEL3 ;
  INCLUDE  PRESBURGER. ARRAY ;
  OP     QS     : array, integer, integer -> array,
         SPRIT : arrintint, integer -> arrintint,
         LEFT, RIGHT : array, integer, integer -> integer ;
  VAR    x        SORT array,
         l, h, b, i SORT integer ;
  AX1:   QS(x, l, h) == IF h ≦ l THEN x
           ELSE QS(QS(PR1(SPRIT(<x, l, h>, CONTENT(x, l))),
                    l, PR2(SPRIT(<x, l, h>, CONTENT(x, l)))),
                PR3(SPRIT(<x, l, h>, CONTENT(x, l))), h) ;
  AX2:   SPRIT(<x, l, h>, b) == IF h ≦ l THEN <x, h, l>
           ELSE IF RIGHT(x, h, b) < LEFT(x, l, b)
                THEN <x, RIGHT(x, h, b), LEFT(x, l, b)>
           ELSE SPRIT(<EXCH(x, RIGHT(x, h, b), LEFT(x, l, b)),
                  LEFT(x, l, b)+1, RIGHT(x, h, b)-1>, b) ;
  LM1:   [ l ≦ h AND CONTENT(x, h) ≧ b ] ⊃
           [ l ≦ LEFT(x, l, b) AND LEFT(x, l, b) ≦ h AND
             b ≦ CONTENT(x, LEFT(x, l, b)) AND
             [ l ≦ i AND i < LEFT(x, l, b) ⊃ CONTENT(x, i) < b ] ]
           == TRUE ;
  LM2:   [ l ≦ h AND CONTENT(x, l) ≦ b ] ⊃
           [ l ≦ RIGHT(x, h, b) AND RIGHT(x, h, b) ≦ h AND
             CONTENT(x, RIGHT(x, h, b)) ≦ b AND
             [ RIGHT(x, h, b) < i AND i ≦ h ⊃ CONTENT(x, i) > b ] ]
           == TRUE ;
  LM3:   [ l < h AND b = CONTENT(x, l) AND
             RIGHT(x, h, b) < LEFT(x, l, b) ] ⊃
           [ l < LEFT(x, l, b) AND RIGHT(x, h, b) < h ] == TRUE ;
END ;
```

表５（レベル３）は，表４（レベル２）のSPRIT に関する７個の性質を証明するのに必要と思われる３個の性質を記述した（表５公理LM1～LM3）．

１．４．４．２　検証の方針

レベル２がレベル１の実現であることの証明は，ソーティングQS(x, l, h) の範囲k(=h-l+1) についての帰納法で証明する．すなわちk-1 個以下の要素のクイックソートに対して，レベル１の(SP1)から(SP3) の３個の性質が成立すること（帰納法の仮定）及び表４（レベル２）のTH1からTH7の公理が成立することを用いて，k個の要素のクイックソートに対し，(SP1)から(SP3)が成立することを証明する．

レベル３がレベル２の実現であることの証明は，表５（レベル３）の公理AX2 及び表１のif関数の公理を用いて，関数SPRIT が表６(1)のように表せること，及び証明すべき各定理（表４の公理TH1 ～TH7 に相当）が，それぞれ適当な述語Qi（i=1, 2, ..., 7）を用いて，Qi(<x, l, h>, SPRIT(<x, l, h>, CONTENT(x, l))) $\cong_E$ TRUE と表せることより，１．３．６で述べた証明法を用いる．例えば，TH1 の証明は，表６(2) の関数Q1を用いて，Q1 $\cong_E$ TRUEと表せる．よってQ1及び表６(3) の関数R1が次の３条件を満たすことを示すことにより証明する．

［条件１］Q1及びR1が次の(i)から(iii)を満たす

（但し，α はCONTENT(x, l)を表す）．

(i)　R1(<x, l, h>, <x, l, h>) $\cong_E$ TRUE

(ii)　[{R1(<x, l, h>, <x′, l′, h′>) AND p(<x′, l′, h′>, α)}
⊃ Q1(<x, l, h>, k(<x′, l′, h′>, α))] $\cong_E$ TRUE

(iii)　[{R1(<x, l, h>, <x′, l′, h′>) AND NOT(p(<x′, l′, h′>, α))}
⊃ R1(<x, l, h>, g(<x′, l′, h′>, α))] $\cong_E$ TRUE

［条件２］ソートarray, integer, integer の任意の構成子項X，L，Hに対し，(i) p(g^n(<X, L, H>, β), β) $\equiv_E$ TRUE，(ii) $0 \leq i < n$ なる任意のｉについて，p(g^i(<X, L, H>, β), β) $\equiv_E$ FALSEを満たすｎが存在する（但し，β はCONTENT(X, L)を表す）．

表6　Q1 ≅ TRUE の証明

```
(1)   SPRIT(<x, l, h>, b)
        ≅ IF p(<x, l, h>, b) THEN k(<x, l, h>, b)
                             ELSE SPRIT(g(<x, l, h>, b), b)
  (但し
      p(<x, l, h>, b)
        = IF h ≦ l THEN TRUE ELSE RIGHT(x, h, b) < LEFT(x, l, b)

      k(<x, l, h>, b)
        = IF h ≦ l THEN <x, h, l>
                   ELSE <x, RIGHT(x, h, b), LEFT(x, l, b)>

      g(<x, l, h>, b)
        = <EXCH(x, RIGHT(x, h, b), LEFT(x, l, b)),
           LEFT(x, l, b)+1, RIGHT(x, h, b)-1>
                                                    とする)
(2)   Q1(<x, l, h>, <x', l', h'>)
        = l < h ⊃ l < h'

(3)   R1(<x, l, h>, <x', l', h'>)
        = l ≦ h ⊃
           [ (l = l' AND h' = h AND CONTENT(x', l') = CONTENT(x, l))
            OR {l < l' AND h' < h AND CONTENT(x', l) ≦ CONTENT(x, l)
                AND CONTENT(x', h) ≧ CONTENT(x, l)} ]
(4)   R(<x, l, h>, <x', l', h'>, i, j, k, v)
        = l ≦ h ⊃ [ {(l < l' AND h' < h) OR
            (l = l' AND h' = h AND CONTENT(x', l') = CONTENT(x, l))}
           AND NUM(x', l, h, v) = NUM(x, l, h, v)
           AND {(k < l OR h < k) ⊃ CONTENT(x', k) = CONTENT(x, k)}
           AND {(l ≦ i AND i < l') ⊃ CONTENT(x', i) ≦ CONTENT(x, l)}
           AND {(h' < j AND j ≦ h) ⊃ CONTENT(x', j) ≧ CONTENT(x, l)} ]
```

［条件３］　p(<X, L, H>, B) $\underset{E}{\equiv}$ FALSE を満たす任意の構成子項Ｘ，Ｌ，Ｈ，Ｂに対し，g(<X, L, H>, B)の値が定義されている．

支援系を用いた条件１の証明については，次節で述べる．尚，一般に条件２，３の証明については，本支援系が直接対象としている等式論理の世界を越える議論を必要とし，検証者が個別に証明しなければならない[13]．又，表６(4) の述語Ｒは，Qi $\underset{E}{\approx}$ TRUE（i=1, 2, ... , 7）の証明に用いた述語Riの論理積をとったものである．

同様に，レベル４がレベル３の実現であることの証明も，１．３．６で述べた証明法を用いる（詳細略）．

１．４．４．３　支援系を用いた証明の例

前節の後半で述べた条件１の証明を例に，本支援系をどのように利用して証明を行うかについて述べる．

［公理の入力］まず，表５の公理系及び関数p, k, g, Q1, R1 の定義式（表６）を公理として（p=t を公理p==tとして）入力する．次に，表７(A) のように，条件１の(i)～(iii)の定理の左辺の項を支援系に入力する．表７では，これらの項をそれぞれterm1, term2, term3とし，以下本文ではそれぞれt1, t2, t3で表す．

［(i) の証明］項t1の標準項t1↓を求め，次にt1↓がプレスブルガー文真偽判定機能を用いてTRUEになることを確認する（表７(B)(*1)）．支援系では，これらの操作をコマンドPRES:term1を用いて行う（100 は，標準項を求めるまでの書き換え回数の上限の指定）．

［(ii)の証明］まず，表７(c) のように，（イ）h′≦l′，（ロ）RIGHT(x′, h′, CONTENT(x, l))<LEFT(x′, l′, CONTENT(x, l))を場合分けの項として選び，４通りの各場合についてt2の標準項t2↓がそれぞれプレスブルガー文真偽判定機能を用いてTRUEになるか否かを確認する（表７(D)）．この時，（イ）がFALSE で（ロ）がTRUEの場合，TRUEと判定されない（表７(D)(*2)）．しかし，プレスブルガー文真偽判定機能を用いて直接 t $\underset{E}{\approx}$ TRUEが証明できない場合でも，既に $\tau \underset{E}{\approx}$ TRUEが示されている項 τ に対し，$\tau \supset$ t $\underset{E}{\approx}$ TRUEが証明できる場

表7　支援系を用いた条件１の証明

```
(A)  command? it:1:2:3
     term1? R1(<X, L, H>, <X, L, H>)
     term2? R1(<X, L, H>, <X', L', H'>) AND p(<X', L', H'>,
            CONTENT(X, L)) ⊃ Q1(<X, L, H>, k(<X', L', H'>,
            CONTENT(X, L)))
     term3?  (省略)

(B)  command? PRES:term1:100
     term1 ——> TRUE :                                    (*1)

(C)  command? ih:1:2
     hypo1? H' ≦ L'
     hypo2? RIGHT(X', H', CONTENT(X, L)) < LEFT(X', L', CONTENT(X, L))

(D)  command? auto:term2:100:PRES:hypo=:**
     hypo1:TRUE  hypo2:TRUE  : term2 ——> TRUE
     hypo1:TRUE  hypo2:FALSE : term2 ——> TRUE
     hypo1:FALSE hypo2:TRUE  : term2 ——> NOT PROVED      (*2)
     hypo1:FALSE hypo2:FALSE : term2 ——> TRUE

(E)  command? auto:term2:100:PRES:hypo=:ft:lemma=:2
     lemma? LM1                                          (*3)
     input instance? x:X':l:L':h:H:b:CONTENT(X, L)       (*4)
     lemma? LM3                                          (*5)
     input instance? x:X':l:L':h:H':b:CONTENT(X, L)      (*6)
     hypo1:FALSE hypo2:TRUE  : term2 ——> TRUE            (*7)
```

合，t ≅TRUEが成立する[13]．特に，公理系Eがτ==TRUEの形の公理を含む場合，変数に対する任意の代入σに対し，σ(τ)≅TRUEが成立する．本支援系では，τ==TRUEなる公理と，τの変数に対する代入σが指定されると，証明すべき定理t ≅TRUEに対し，項σ(τ)⊃tを自動的に作成し，プレスブルガー文真偽判定機能を用いて，この項の真偽を判定する機能を持つ（公理が複数個指定された時は，σ1(τ1)⊃σ2(τ2)･･･σn(τn)⊃tを作成する）．上述のTRUEと判定されない場合，この機能を用いて，表7 (E) (*3), (*5)のようにτ==TRUEの形の公理LM1とLM3を指定し，又，(*4) (*6)のように，各変数に対する代入σ1，σ2を指定して（τ1，τ2はそれぞれLM1，LM3の左辺），項σ1(τ1)⊃σ2(τ2)⊃t2↓がTRUEになることを確認した（表7 (E) (*7)）．

［(iii) の証明］(iii) の証明は，(ii)と同様の方法で行った[13]．

1．5　結言

筆者らは，従来の項書き換えのみによる方法と，プレスブルガー文の真偽判定機能も利用する方法で幾つかの検証を行った．その結果，例えば，1．4で述べた表4（レベル2）が表3（レベル1）の実現であることの証明では，検証に用いた補題の個数がそれぞれ43個と，22個，検証に要した計算機との対話時間がそれぞれ19時間と10時間であった．又，Stenningのプロトコルの検証では，補題の個数が52個と10個，対話時間は20時間と8時間であった．尚，プレスブルガー文の真偽判定機能を用いた場合に必要な補題は，いわゆる証明の"ロジック"に関するもののみであり，単にブールや整数の性質に関する補題は不要であった．

第２章　代数的に記述されたHDLCプロトコルの検証

２．１　序言

伝送制御手順（プロトコル）の検証のためには，手順自身が厳密に記述されていなければならない．従来の記述法は，状態遷移図を利用する方法[14]と，プログラム言語を利用する方法[7]とに大別される．前者の記述法を用いた時の検証における問題点は，プロトコルの複雑化に伴い，状態数が急激に増加することであり，後者の問題点は，処理順序の一意的記述により過剰規定に陥り易いことや，利用する言語の意味の形式的定義が複雑であることである．

代数的仕様記述法[8]は，これらの欠点を免れ得る有効な手法の１つであると考えられる．この章では，プロトコルの例としてハイレベルデータリンク制御（HDLC）手順[17]を選び，代数的手法による検証結果について報告する．

筆者らは，まずHDLC手順の規格[17]を分析し，その結果に基づき，HDLC手順を用いて通信を行う通信系を抽象的な順序機械と考え代数的記述を行った．これらの記述は，(1) HDLC手順の転送単位であるフレームの代数的記述，(2)，(3) １，２次局の代数的記述，(4) 物理レベルの代数的記述，及び (5) これら４個の記述を基に通信系内の相互関係（いわゆるインターフェイス部分）を記述した通信系全体の代数的記述より構成される．次に，これらの記述の上で通信系全体での動作可能性や，１，２次局の状態変数間に成立する関係など２．６で述べるような幾つかの検証を行った．検証は，主に構造的帰納法を用いて行い，検証作業の効率化と確実性を向上させるため，第１章で述べた代数的仕様検証支援系を利用した．

２．２　諸定義

以下，基本的定義は１．２の定義に従う．尚，この章では表８のような基底代数の仕様DB=(ＳB, ΣB, ＥB)を部分仕様として用いる．

(1) ＳＢはブール，非負整数，データの集合を表すソートbool, nn-integer, data より構成される．

(2) " TRUE" (真)，" FALSE " (偽)，" 0 " ，" 1 " ，・・・(非負整数) をΣＢの構成子として，" AND " (論理積)，" OR" (論理和)，" NOT " (否定)，" ＋" (加算)，" －" (減算)，" ＝" (等号)，" ≧" ，" ＞" ，" ≦" ，" ＜" (大小比較)，" MOD " (モード演算)をΣＢの定義関数とする．

(3) 公理系ＥＢ (公理は有限個とは限らない) は，ΣＢのすべての定義関数ｆ及びｆの各引数ソートの任意の構成子項tiの組に対して，" f(t1, t2, ..., tn)==t" (但し，ｔはｆの値域ソートの構成子項) の形の公理から成るとする．すなわち，ＥＢはΣＢの定義関数に対する関数表と考えることができる．

又，すべてのソートｓについて，次のような公理を持つ定義関数" ifs:bool, s, s->s " ，及び" if-thens:bool, s->s" を用いる．

ifs(TRUE, x, y) == x ;

ifs(FALSE, x, y) == y ;

if-thens(TRUE, x) == x ;

(但し，ｘ，ｙはソートｓの変数とする) 以下，混同がない限り，ifs(.....), if-thens(.....) をそれぞれ" IF...THEN... ELSE... " ，" IF...THEN..." で表し，添字ｓは省略する．又，" IF t THEN t′ ELSE TRUE" を" t ⊃ t′" と略記する．

表8　基 底 代 数

```
SPEC   PRESBURGER :
  SORT bool, nn-integer, data :
  CON  TRUE, FALSE   : -> bool
       0, 1, 2, ...  : -> nn-integer :
  OP   AND, OR : bool, bool -> bool,
       NOT     : bool       -> bool,
       +, -, MOD      : nn-integer, nn-integer -> nn-integer,
       =, >, ≧, <, ≦ : nn-integer, nn-integer -> bool :
EMD :


SPEC   E. PRESBURGER (m) :
  INCLUDE  PRESBURGER :
  OP   plus1 : nn-integer -> nn-integer,
       order : nn-integer, nn-integer, nn-integer -> bool :
  VAR  m, s, t, u SORT nn-integer :
  EP1: plus1 (s) == (s+1) MOD m :
  EP2: order (s, t, u) == [ (s-u) MOD m ] ≧ [ (t-u) MOD m ] :
END :
```

２．３　通信系全体の代数的記述

HDLC手順には，システム構成や動作モードの違いなどにより幾つかの手順クラスが規定されているが，ここでは標準的な手順クラスである不平衡型正規応答モード（クラスUN）について記述する．又，この章では，１，２次局間で通信が行われている時に成り立つ状態変数間の論理関係式の検証などを主な検証の対象とし，以下ではそれらの検証に必要かつ十分な抽象レベルでそれぞれの記述を行い，検証に直接不必要な部分（データリンクの設定，終結部分，フレーム，物理レベルの詳細など）の記述は省略した．

２．３．１　フレーム及び１次局，２次局の代数的記述

フレームの代数的記述（表９）ではフレームの集合を表すソートをframe とする．情報(I) フレーム，RRフレーム，RNR フレームを構成する関数（構成子）をそれぞれIframe, RRframe, RNRframe とし，各フレームがＩフレームかRRフレームかRNR フレームか否かを判定する関数をそれぞれtypeI, typeRR, typeRNRで表す．又，各フレームから送信順序番号，受信順序番号，P/F ビット，情報部のデータを取り出す関数（定義関数）をそれぞれssn, rsn, pfbit, informとした．

各情報フレームは，０から”モジュラス－１”までの値の順序番号を持つ．順序番号は，この範囲内のすべての数を使用して，連続的に循環する．ここで，モジュラスとは順序番号の法を指し，通常は８であるが，拡張モードでは128 である．本論文ではモジュラスの値が８及び128 のいずれでも可能なように，モジュラスの値をパラメータMODULUS として記述している．

１次局，２次局については，それぞれ内部構造を陽に持たない抽象的”状態”を導入し（導入した状態を表すソートをそれぞれstate.P, state.S と表す），送信，受信，タイムアウトの発生といった基本動作を”状態遷移関数”（構成子）とし，各局の送信状態変数などは”状態”からの”出力関数”（定義関数）として導入した．又，それぞれの状態遷移後における各出力関数の値が遷移前の

表9　フレームの代数的記述

```
SPEC    FRAME ;
  INCLUDE  E. PRESBURGER (MODULUS) ;
  SORT  frame ;
  CON   Iframe : nn-integer, nn-integer, data, bool -> frame,
        RRframe, RNRframe: nn-integer, bool -> frame ;
  OP    typeI, typeRR, typeRNR : frame -> bool,
        ssn, rsn               : frame -> nn-integer,
        pfbit                  : frame -> bool,
        inform                 : frame -> data ;
  OP    MODULUS                : -> nn-integer ;
  VAR   s, r SORT nn-integer,
        d    SORT data,
        b    SORT bool ;
  FR1:  typeI (Iframe (s, r, d, b)) == TRUE ;
  FR2:  typeI (RRframe (r, b)) == FALSE ;
  FR3:  typeI (RNRframe (r, b)) == FALSE ;
  FR4:  typeRR (Iframe (s, r, d, b)) == FALSE ;
  FR5:  typeRR (RRframe (r, b)) == TRUE ;
  FR6:  typeRR (RNRframe (r, b)) == FALSE ;
  FR7:  typeRNR (Iframe (s, r, d, b)) == FALSE ;
  FR8:  typeRNR (RRframe (r, b)) == FALSE ;
  FR9:  typeRNR (RNRframe (r, b)) == TRUE ;
  FR10: ssn (Iframe (s, r, d, b)) == s ;
  FR11: rsn (Iframe (s, r, d, b)) == r ;
  FR12: rsn (RRframe (r, b)) == r ;
  FR13: rsn (RNRframe (r, b)) == r ;
  FR14: pfbit (Iframe (s, r, d, b)) == b ;
  FR15: pfbit (RRframe (r, b)) == b ;
  FR16: pfbit (RNRframe (r, b)) == b ;
  FR17: inform (Iframe (s, r, d, b)) == d ;
END ;
```

状態における出力関数の値からどのように定まるかを公理として表した．表10，表11はそれぞれ１次局，２次局の代数的記述である．尚，表12は１次局，２次局の共通部分の仕様であり，表10，表11の仕様ではそれぞれ表12の仕様を部分仕様として利用している†．

たとえば，１次局の記述ではＩフレーム，RRフレーム，RNR フレームの送信を表す関数SendI.P，SendRR.P，SendRNR.P，フレームの受信を表す関数Receive.P，タイムアウトの発生を表す関数Timeout.P を状態遷移関数として導入し，次のような関数を出力関数として導入した．

(1) 送信状態変数を表す関数: seq.P，(2) 受信状態変数を表す関数: ack.P，(3) 最旧未確認状態変数[18]を表す関数: low.P，(4) P/F ビット送信時状態変数[18]を表す関数: checkpoint.P，(5) P/F ビットが"１"のフレームを送信出来るか否かを表す関数: ready.P．

又，規格で許されない動作の条件をひとまとめにして表す関数（定義関数）valid.P を導入し，初期状態から規格で許される動作のみを行って到達した状態に対してのみ，valid.P の値がブール値"真"になるように公理が書かれている．

２次局の記述については，タイムアウトを発生させる関数がない事を除いて１次局の記述とほぼ同様であり，上述の各関数の".P"を".S"で置き換えた関数をそれぞれ２次局の関数として用いる．

---

† 記述を簡単にするため表８のE.PRESBURGERで宣言した２つの補助関数plus1，orderを用いる（表８のEP1，EP2参照）．又，表12の公理 C08* のように，公理のラベルに＊印の付いているものは次のような略記法を用いることを表している（表10～表12）．

略記法: "ラベル*: f(g(...),...) == t ;"の形の公理は，"ラベル: f(g(...),...) == IF valid.x(g(...)) THEN t ;"を表すとする．

表10　1次局の代数的記述

```
SPEC    PRIMARY ;
  INCLUDE  COMMON(.P) ;
  CON    Timeout.P       : state.P -> state.P ;
  OP     timeouterror.P : stata.P -> bool ;
  VAR    p SORT state.P,
         f SORT frame,
         b SORT bool,
         n SORT nn-integer ;
  PR1 : timeouterror.P(Initial.P) == FALSE ;
  PR2 : ready.P(Initial.P) == TRUE ;
  PR3*: timeouterror.P(SendI.P(p,b)) == timeouterror.P(p) ;
  PR4 : valid.P(SendI.P(p,b)) == IF valid.P(p)
         THEN NOT(plus1(seq.P(p))=low.P(p)) AND (ready.P(p) OR NOT(b))
           AND NOT(timeouterror.P(p)) AND NOT(busy.P(p))
         ELSE FALSE ;
  PR5*: timeouterror.P(SendRR.P(p,b))
         == NOT(b) AND timeouterror.P(p) ;
  PR6 : valid.P(SendRR.P(p,b)) == IF valid.P(p)
         THEN ready.P(p) OR NOT(b) ELSE FALSE ;
  PR7*: timeouterror.P(SendRNR.P(p,b)) == timeouterror.P(p)
  PR8 : valid.P(SendRNR.P(p,b)) == IF valid.P(p)
         THEN (ready.P(p) OR NOT(b)) AND NOT(timeouterror.P(p))
         ELSE FALSE ;
  PR9*: timeouterror.P(Receive.P(p,f)) == timeouterror.P(p) ;
  PR10*: seq.P(Timeout.P(p)) == seq.P(p) ;
  PR11*: high.P(Timeout.P(p)) == high.P(p) ;
  PR12*: low.P(Timeout.P(p)) == low.P(p) ;
  PR13*: ack.P(Timeout.P(p)) == ack.P(p) ;
  PR14*: checkpoint.P(Timeout.P(p)) == checkpoint.P(p) ;
  PR15*: timeouterror.P(Timeout.P(p)) == TRUE ;
  PR16*: ready.P(Timeout.P(p)) == TRUE ;
  PR17*: busy.P(Timeout.P(p)) == busy.P(p) ;
  PR18 : valid.P(Timeout.P(p)) == valid.P(p) ;
END ;
```

表11　2次局の代数的記述

```
SPEC    SECONDARY ;
  INCLUDE  COMMON(.S) ;
  VAR   p SORT state.S,
        f SORT frame,
        b SORT bool,
        n SORT nn-integer ;
  SE1*: ready.S(Initial.S) == FALSE ;
  SE2 : valid.S(SendI.S(p,b)) == IF valid.S(p)
         THEN NOT(plus1(seq.S(p)) = low.S(p))
           AND (ready.S(p) AND NOT(busy.S(p)))
         ELSE FALSE ;
  SE3 : valid.S(SendRR.S(p,b)) == IF valid.S(p)
         THEN ready.S(p)  ELSE FALSE ;
  SE4 : valid.S(SendRNR.S(p,b)) == IF valid.S(p)
         THEN ready.S(p)  ELSE FALSE ;
END ;
```

表12　1次局と2次局の共通部分の代数的記述

```
SPEC    COMMON(.x) ;
  INCLUDE  FRAME ;
  SORT  state.x ;
  CON   Initial.x ;                  -> state.x,
        SendI.x, SendRR.x, SendRNR.x
                  : state.x, bool    -> state.x,
        Receive.x : state.x, frame   -> state.x ;
  OP    seq.x, high.x, low.x, ack.x, checkpoint.x
                  : state.x          -> nn-integer,
        busy.x, ready.x, valid.x
                  : state.x          -> bool ;
  VAR   p SORT state.x,
        f SORT frame,
        b SORT bool,
        n SORT nn-integer ;
  CO1 : seq.x(Initial.x) == 0 ;
  CO2 : high.x(Initial.x) == 0 ;
  CO3 : low.x(Initial.x) == 0 ;
  CO4 : ack.x(Initial.x) == 0 ;
  CO5 : checkpoint.x(Initial.x) == 0 ;
  CO6 : busy.x(Initial.x) == FALSE ;
  CO7 : valid.x(Initial.x) == TRUE ;
```

(次ページへ続く)

表12　1次局と2次局の共通部分の代数的記述

(前ページより続く)

```
C08*: seq.x(SendI.x(p,b)) == plus1(seq.x(p)) ;
C09*: high.x(SendI.x(p,b)) == IF seq.x(p) = high.x(p)
        THEN plus1(high.x(p))  ELSE high.x(p) ;
C010*: low.x(SendI.x(p,b)) == low.x(p) ;
C011*: ack.x(SendI.x(p,b)) == ack.x(p) ;
C012*: checkpoint.x(SendI.x(p,b)) == IF b
        THEN seq.x(p)  ELSE checkpoint.x(p) ;
C013*: busy.x(SendI.x(p,b)) == busy.x(p) ;
C014*: ready.x(SendI.x(p,b)) == NOT(b) AND ready.x(p) ;
C015*: seq.x(SendRR.x(p,b)) == seq.x(p) ;
C016*: high.x(SendRR.x(p,b)) == high.x(p) ;
C017*: low.x(SendRR.x(p,b)) == low.x(p) ;
C018*: ack.x(SendRR.x(p,b)) == ack.x(p) ;
C019*: checkpoint.x(SendRR.x(p,b)) == IF b
        THEN seq.x(p)  ELSE checkpoint.x(p) ;
C020*: busy.x(SendRR.x(p,b)) == busy.x(p) ;
C021*: ready.x(SendRR.x(p,b)) == NOT(b) AND ready.x(p) ;
C022*: seq.x(SendRNR.x(p,b)) == seq.x(p) ;
C023*: high.x(SendRNR.x(p,b)) == high.x(p) ;
C024*: low.x(SendRNR.x(p,b)) == low.x(p) ;
C025*: ack.x(SendRNR.x(p,b)) == ack.x(p) ;
C026*: checkpoint.x(SendRNR.x(p,b)) == IF b
        THEN seq.x(p)  ELSE checkpoint.x(p) ;
C027*: busy.x(SendRNR.x(p,b)) == busy.x(p) ;
C028*: ready.x(SendRNR.x(p,b)) == NOT(b) AND ready.x(p) ;
C029*: seq.x(Receive.x(p,f)) == IF pfbit(f)
           AND order(checkpoint.x(p),rsn(f),low.x(p))
           AND NOT(checkpoint.x(p) = rsn(f))
        THEN rsn(f)  ELSE seq.x(p) ;
C030*: high.x(Receive.x(p,f)) == high.x(p) ;
C031*: low.x(Receive.x(p,f)) == rsn(f) ;
C032*: ack.x(Receive.x(p,f)) == IF typeI(f)
        THEN IF ssn(f) = ack.x(p)
             THEN plus1(ack.x(p))  ELSE ack.x(p)
        ELSE ack.x(p) ;
C033*: checkpoint.x(Receive.x(p,f))
        == IF order(rsn(f),checkpoint.x(p),low.x(p))
           THEN rsn(f)  ELSE checkpoint.x(p) ;
C034*: busy.x(Receive.x(p,f)) == IF typeRR(f) OR
             (typeI(f) AND pfbit(f))
        THEN FALSE  ELSE typeRNR(f) OR busy.x(p) ;
C035*: ready.x(Receive.x(p,f)) == pfbit(f) OR ready.x(p) ;
C036 : valid.x(Receive.x(p,f)) == valid.x(p) ;
END ;
```

2．3．2　物理レベルの代数的記述

往路（1次局から2次局）及び復路（2次局から1次局）の2つの伝送路を含む物理レベルの系を以下のように記述する（表13，表14）．

往路の伝送路の記述（表13）においては，1，2次局の記述と同様に，伝送路の抽象的”状態”を表すソートstate.F を導入し，次のような関数を状態遷移関数（構成子）として導入した．

(1) 初期状態を表す関数: Initial.F ，(2) 指定されたフレームを伝送路の後端に追加することを表す関数: Add.F ，(3) 伝送路の先端から1つのフレームを消去することを表す関数: Delete.F.

Add.F は，1次局のフレームの送信に対応して用いられる関数であり，Delete.Fは，2次局のフレームの受信や，伝送路上でのフレームの消失に対応する関数である（フレームの消失は伝送路の先頭で起こると考えても外への影響は同一である）．尚，受信側でフレームの伝送誤りを検出した時の状態変数等の値は，そのフレームが消失した時の場合と同一であるので[17]，両者を同一視して扱い，伝送路上のフレームの伝送誤りを引き起す関数は使用しないことにする．

又，伝送路上で許されない動作（伝送路上にフレームが1つも乗っていない時にDelete.Fを実行すること）が行われたか否かを表す関数valid.F （2．3．1のvalid.P に相当）及び，伝送路上に乗っているフレームの数を表す関数count.F 及び伝送路の先頭からi番目のフレームの内容を表す関数access.Fを出力関数（定義関数）として導入し，それぞれの状態遷移の前後での出力関数間の関係を公理として記述した．

復路の伝送路の記述（表14）は，往路の伝送路の記述と同一であり表13の各関数の”.F”を”.B”で置き換えたものとする．

表13　往路の伝送路の代数的記述

```
SPEC    FLINE ;
  INCLUDE  FRAME ;
  SORT  state.F ;
  CON   Initial.F :                          -> state.F,
        Add.F     : state.F, frame           -> state.F,
        Delete.F  : state.F                  -> state.F ;
  OP    count.F   : state.F                  -> nn-integer,
        access.F  : state.F, nn-integer      -> frame,
        valid.F   : state.F                  -> bool ;
  VAR   f SORT frame,
        p SORT state.F,
        i SORT nn-integer ;
  FL1:  count.F(Initial.F) == 0 ;
  FL2:  valid.F(Initial.F) == TRUE ;
  FL3:  count.F(Add.F(p, f)) == count.F(p)+1 ;
  FL4:  access.F(Add.F(p, f), i) ==
          IF i = count.F(p)+1  THEN f
          ELSE IF count.F(p) ≧ i AND NOT(i = 0)
               THEN access.F(p, i) ;
  FL5:  valid.F(Add.F(p, f)) == valid.F(p) ;
  FL6:  count.F(Delete.F(p)) == count.F(p)-1 ;
  FL7:  access.F(Delete.F(p), i) ==
          IF count.F(p)-1 ≧ i AND NOT(i = 0)
             THEN access.F(p, i+1) ;
  FL8:  valid.F(Delete.F(p)) == IF valid.F(p)
          THEN NOT(count.F(p) = 0)  ELSE FALSE ;
END ;
```

表14　復路の伝送路の代数的記述

```
SPEC    BLINE ;
  INCLUDE  FRAME ;
  SORT  state.B ;
  CON   Initial.B :                          -> state.B
        Add.B     : state.B  frame           -> state.B
        Delete.B  : state.B                  -> state.B ;
  OP    count.B   : state.B                  -> nn-integer,
        access.B  : state.B  nn-integer      -> frame,
        valid.B   : state.B                  -> bool ;
  VAR   f SORT frame,
        p SORT state.B
        i SORT nn-integer ;
  BL1:  count.B(Initial.B) == 0 ;
  BL2:  valid.B(Initial.B) == TRUE ;
  BL3:  count.B(Add.B(p, f)) == count.B(p)+1 ;
  BL4:  access.B(Add.B(p, f), i) ==
          IF i = count.B(p)+1  THEN f
          ELSE IF count.B(p) ≧ i AND NOT(i = 0)
               THEN access.B(p, i) ;
  BL5:  valid.B(Add.B(p, f)) == valid.B(p) ;
  BL6:  count.B(Delete.B(p)) == count.B(p)-1 ;
  BL7:  access.B(Delete.B(p), i) ==
          IF count.B(p)-1 ≧ i AND NOT(i = 0)
             THEN access.B(p, i+1) ;
  BL8:  valid.B(Delete.B(p)) == IF valid.B(p)
          THEN NOT(count.B(p) = 0)  ELSE FALSE ;
END ;
```

2．3．3　通信系全体の代数的記述

これまでは，1次局，2次局及び伝送路についてそれぞれ独立した記述を行ってきた．しかし通信系全体においては，1次局からのIフレームの送信は，1次局でのフレームの送信という状態遷移（SendI.P の実行に相当）と，往路の伝送路へのフレームの追加（Add.F の実行に相当）という状態遷移が，共に実行されることに相当する．このような相互関係を表すため，1次局，2次局，往路及び復路の伝送路の4つの記述を部分仕様として含む通信系全体の代数的記述を作成した（表15）．この記述では，通信系全体の”状態”を表すソートをstate とし，ソートstate を上述の4つの部分仕様の状態を表すソートstate.P, state.S, state.F 及びstate.B の”直積”と考えている．そして，1次局，2次局の各基本動作や伝送路上でのフレームの消失など通信系全体の状態変化を引き起こす動作をソートstate の状態遷移関数（構成子）として導入し，これらの状態遷移関数の実行がソートstate.P, state.S, state.F 及びstate.B におけるどのような状態変化に対応するかの関係を公理として表した．

たとえば，上述の1次局からのIフレームの送信を表すソートstate の状態遷移関数をTsendI.Pと書き，表15の4つの公理（T011～T014）によって，TsendI.P(t, d, b)の実行が，1次局でのSendI.P(prim(t), b)の実行と往路の伝送路でのAdd.F(fline(t), Iframe(..))の実行（1次局の送信状態変数などからIフレームを構成し，伝送路の後端に追加する事を表す）からなる操作であることを表している．ここで，関数prim, secon, fline及びbline は通信系全体の状態state から上述の4つの部分仕様における状態ソートの成分を取り出す定義関数である．

上述のTsendI.P以外の状態遷移関数として次のような関数を導入した．

(1) 通信系全体の初期状態を表す関数: Tinitial, (2) 1次局から2次局へRR, RNRフレームを送信する関数: TsendRR.P, TsendRNR.P,

表15　通信系全体の代数的記述

```
SPEC    TOTAL :
  INCLUDE  PRIMARY, SECONDARY, FLINE, BLINE :
  SORT  state :
  CON   Tinitial :                          -> state,
        TsendI.P, TsendI.S
                 : state, data, bool        -> state,
        TsendRR.P, TsendRNR.P, TsendRR.S, TsendRNR.S
                 : state, bool              -> state,
        Treceive.P, Ttimeout.P, Treceive.S, Tdrop.F, Tdrop.B
                 : state                    -> state :
  OP    prim     : state                    -> state.P,
        secon    : state                    -> state.S,
        fline    : state                    -> state.F,
        bline    : state                    -> state.B,
        valid    : state                    -> bool :
  VAR   t SORT state,
        d SORT data,
        b SORT bool :
  T01:  prim(Tinitial) == Initial.P :
  T02:  secon(Tinitial) == Initial.S :
  T03:  fline(Tinitial) == Initial.F :
  T04:  bline(Tinitial) == Initial.B :
  T011: prim(TsendI.P(t, d, b)) == SendI.P(prim(t), b) :
  T012: secon(TsendI.P(t, d, b)) == secon(t) :
  T013: fline(TsendI.P(t, d, b)) == Add.F(fline(t),
          Iframe(seq.P(prim(t)), ack.P(prim(t)), d, b)) :
  T014: bline(TsendI.P(t, d, b)) == bline(t) :
  T021: prim(TsendRR.P(t, b)) == SendRR.P(prim(t), b) :
  T022: secon(TsendRR.P(t, b)) == secon(t) :
  T023: fline(TsendRR.P(t, b)) == Add.F(fline(t),
          RRframe(ack.P(prim(t)), b)) :
  T024: bline(TsendRR.P(t, b)) == bline(t) :
  T031: prim(TsendRNR.P(t, b)) == SendRNR.P(prim(t), b) :
  T032: secon(TsendRNR.P(t, b)) == secon(t) :
  T033: fline(TsendRNR.P(t, b)) == Add.F(fline(t),
          RNRframe(ack.P(prim(t)), b)) :
  T034: bline(TsendRNR.P(t, b)) == bline(t) :
  T041: prim(Treceive.P(t)) == Receive.P(prim(t),
          access.B(bline(t), 1)) :
  T042: secon(Treceive.P(t)) == secon(t) :
  T043: fline(Treceive.P(t)) == fline(t) :
  T044: bline(Treceive.P(t)) == Delete.B(bline(t)) :
  T051: prim(Ttimeout.P(t)) == Timeout.P(prim(t)) :
  T052: secon(Ttimeout.P(t)) == secon(t) :
  T053: fline(Ttimeout.P(t)) == Initial.F :
  T054: bline(Ttimeout.P(t)) == Initial.B :
```

(次ページへ続く)

表15　通信系全体の代数的記述

(前ページから続く)

```
T061: prim(TsendI.S(t,d,b)) == prim(t) ;
T062: secon(TsendI.S(t,d,b)) == SendI.S(secon(t),b) ;
T063: fline(TsendI.S(t,d,b)) == fline(t) ;
T064: bline(TsendI.S(t,d,b)) == Add.B(bline(t),
        Iframe(seq.S(secon(t)),ack.S(secon(t)),d,b)) ;
T071: prim(TsendRR.S(t,b)) == prim(t) ;
T072: secon(TsendRR.S(t,b)) == SendRR.S(secon(t),b) ;
T073: fline(TsendRR.S(t,b)) == fline(t) ;
T074: bline(TsendRR.S(t,b)) == Add.B(bline(t),
        RRframe(ack.S(secon(t)),b)) ;
T081: prim(TsendRNR.S(t,b)) == prim(t) ;
T082: secon(TsendRNR.S(t,b)) == SendRNR.S(secon(t),b) ;
T083: fline(TsendRNR.S(t,b)) == fline(t) ;
T084: bline(TsendRNR.S(t,b)) == Add.B(bline(t),
        RNRframe(ack.S(secon(t)),b)) ;
T091: prim(Treceive.S(t)) == prim(t) ;
T092: secon(Treceive.S(t)) == Receive.S(secon(t),
        access.F(fline(t),1)) ;
T093: fline(Treceive.S(t)) == Delete.F(fline(t)) ;
T094: bline(Treceive.S(t)) == bline(t) ;
DR11: prim(Tdrop.F(t)) == prim(t) ;
DR12: secon(Tdrop.F(t)) == secon(t) ;
DR13: fline(Tdrop.F(t)) == Delete.F(fline(t)) ;
DR14: bline(Tdrop.F(t)) == bline(t) ;
DR21: prim(Tdrop.B(t)) == prim(t) ;
DR22: secon(Tdrop.B(t)) == secon(t) ;
DR23: fline(Tdrop.B(t)) == fline(t) ;
DR24: bline(Tdrop.B(t)) == Delete.B(bline(t)) ;
TV01: valid(t) == valid.P(prim(t)) AND valid.S(secon(t))
        AND valid.F(fline(t)) AND valid.B(bline(t)) ;
END ;
```

(3) 1次局が復路の伝送路からフレームを受信する関数: Treceive.P, (4) 1次局でタイムアウトを起こす関数: Ttimeout.P (以下では，動的な性質の検証は行わないとする．このため，タイマーがいつセットされ，いつ解除されるかなどに関しては，陽に記述せず，タイムアウトの発生によりどのような状態変化が起こるかについてのみ記述した)．(5) 2次局から1次局へI，RR, RNR フレームを送信する関数TsendI.S, TsendRR.S, TsendRNR.S，(6) 2次局が往路の伝送路からフレームを受信する関数: Treceive.S, (7) 往路，復路の伝送路でフレームの消失を引き起こす関数: Tdrop.F, Tdrop.B.

又，通信系全体で許されない動作の条件をひとまとめにして表す定義関数valid を導入した (表15のTV01)．

2．4　通信系全体の代数的記述の性質

通信系全体の代数的記述（表15）の公理系（以下Eで表す）は，次のような性質を持つ．

［性質1（項書き換え系）］(イ)公理系Eは項書き換え系とみなすことが出来る．又，(ロ)R_EはChurch-Rosser である．

（証明）(イ)公理系Eの各公理の右辺に現れる変数は必ず左辺にも現れているので，Eは項書き換え系とみなせる．(ロ)さらに，各公理の左辺には，同一変数が複数回現れず，又，各公理の左辺が他のどの公理の左辺とも重ならず，且つ自分自身のどの真部分項とも重ならない（R_Eはnon-overlapping[9]）．従って，文献（9）から，R_Eは，Church-Rosser である．

［性質2（有限停止性）］公理系Eを項書き換え系とみなす時，R_Eは有限停止性を持つ．

（証明）任意の項 $t \in T(\Sigma+V)$ に対し，項tを木表現し，各葉から根に至る道Piに対して非負整数の3字組wi=<xi, yi, zi> を次のように定める．(イ)xiはPi上に現れる値域又は定義域にソートstateを含む関数の個数，(ロ)yiは値域又は定義域にソートstate.P, state.S, state.F, state.B 又はframe を含む関数の個数，(ハ)ziは基底代数の定義関数の個数．

項tの葉から根に至るすべての道をP1, P2, ..., Pmとする．Piに対応する3字組wiを要素とする多重集合† {w1, ..., wm} をM(t)で表す．又，3字組の辞書式順序" >*" を次のように定義する（但し，wi=<xi, yi, zi>, wj=(xj, yj, zj> とする）．

(i)　xi > xj ならばwi >* wj

(ii)　xi = xj の時，yi > yj ならばwi >* wj

(iii) xi = xj, yi = yj の時，zi > zj ならばwi >* wj

次に $T(\Sigma+V)$ 上の半順序" >>" を次のように定義する．

---

†　同じ3字組を重複して考える．

(1) M(t), M(t′)に含まれる要素を”>*”の順にそれぞれ並び換え，各多重集合の最大要素wmax, w′maxに対して，wmax >* w′max ならば，t >>t′とする．但し，M(t) が空集合でなく，M(t′)が空集合の場合も t >>t′とする．

(2) wmax = w′maxならば，M(t), M(t′)からそれぞれwmax, w′maxを除いた多重集合で同様の比較を行う．

この時，t R_E t′なる任意の項t，t′に対して，t >>t′が成り立つことが示せる．”>>”の定義より明らかにt0>>t1>>・・・なる無限の系列は存在しない（”>>”はwell-foundedである[19]）．従って，R_Eは有限停止性を持つ．

上述の2つの性質より，次の性質が成り立つ．

［性質3（合同関係の判定）］[16]公理系E上での2項の合同関係の判定が，それぞれの項の標準項が一致するか否かで判定出来る．すなわち，任意の項t，t′に対して，t $\equiv_E$ t′<==>t↓=t′↓が成り立つ．

又，構成子が左辺の先頭の関数であるような公理が存在しないことより次の性質が成り立つ．

［性質4（無矛盾性）］[9]公理系Eは無矛盾である．すなわち，同一ソートの異なる2つの構成子項が公理系Eの下で等価になることはない．

又，文献（15）と同様に次の性質が成り立つ．

［性質5（準完全性）］(イ)通信系全体の状態を表すソートstateの任意の構成子項tに対して，valid(t), valid.P(prim(t)), valid.S(secon(t)), valid.F(fline(t)), valid.B(bline(t)) の値がブール値”真”又は”偽”として一意に定まる．又，prim(t), secon(t), fline(t), bline(t)の値がソートstate.P, state.S, state.F, state.B の構成子項として一意に定まる．

(ロ)ソートstate.P, state.S, state.F 又はstate.B のみを定義域ソートとするすべての定義関数fについては，valid(t)の値がブール値”真”である任意の構成子項tに対して，f(prim(t)), f(secon(t)), f(fline(t))又はf(bline(t))の値が基底代数の元（構

成子項）として一意に定まる．valid(t)の値がブール値”真”である任意の構成子項 t 及び 0<N ≦ count.F(fline(t)) を満たす任意の非負整数Nに対し，access.F(fline(t),N)の値がソート frame の構成子項として一意に定まる．access.Bについても同様である．

２．５　通信系全体の公理系上での検証

この章では，プロトコル実行中に成り立つ不変式の証明を主な検証の対象としている．すなわち，検証は通信系全体の公理系Ｅ上で命題" P $\equiv_E$ TRUE" 又は，" P $\cong_E$ TRUE" を証明することと考える．P $\equiv_E$ TRUEの証明は，２．４で述べたようにＰの標準項P↓ がTRUEであることを示すことである．又，Ｐ $\cong_E$ TRUEの証明は１．３で述べた幾つかの方法を用いて行った．以下では，２．６の定理（Ｐ $\cong_E$ TRUE)の証明に用いた主な検証法について説明する（但し，説明を簡単にするためＰに現れる変数がソートstate の変数ｓのみの場合（P(s) $\cong_E$ TRUE) について説明する）．

(1) P(s) $\equiv_E$ TRUEならばP(s) $\cong_E$ TRUEであり，ごく簡単な定理の証明は R_E上でP(s)↓=TRUE を示すことにより検証できた．

(2) 定理の追加による証明

公理系Ｅ上の定理（その定理の正しさはすでに証明済み）のいくつかを公理系Ｅに追加し（拡張された公理系をE1とする），E1上でP(s) $\cong_E$ TRUEを証明した．但し，追加した公理の適用はその変数に代入される項の値が構成子項全域で定義されていることが保証されている時に限った．尚，基底代数上の定理は，その定理の正しさは自明であるとし，検証者の責任において（証明なしで）公理系Ｅに追加した．

(3) 場合分けによる方法

筆者らが行った検証の多くの場合，P(s)↓を求めると，" IF q(s) THEN P1(s) ELSE P2(s) " の形で表現できた．この時，次の(i)，(ii)を示すことによりP(s) $\cong_E$ TRUEの証明を行った（但し，" S " はソートstate の任意の構成子項を表すために導入した定数関数．又，E1=E U {q(S)==TRUE} ，E2=E U {q(S)==FALSE}）．

(i)　P1(S) $\cong_{E1}$ TRUE

(ii)　P2(S) $\cong_{E2}$ TRUE

(4) 構造的帰納法による証明

通常，P(s)≅TRUEの証明はソートstate の構成子項の複合の深さに関する構造的帰納法を用いて行った．すなわち，次の(i),(ii)を示すことにより証明した．

(i) 初期条件の証明：ソートstate の定数関数Tinitialに対して，P(Tinitial)≡TRUE すなわち，P(Tinitial)↓=TRUEを証明する．

(ii)帰納段階の証明：項の深さがn段であるようなソートstate の任意の構成子項を表す特別な定数関数”C”を導入し，帰納法の仮定として，”P(C)==TRUE”を公理系Eに追加する（その公理系をE1とする）．次に，ソートstate を定義域ソートに含み，且つソートstate を値域ソートとするすべての構成子gjに対して，P(gj(C,y1,y2,....,ym))≅TRUE を証明する（但し，各yiは基底代数の変数である）．

尚，y1,y2,....,ymについては，帰納法を適用するよりも，(2) で述べたように検証上有用と思われる基底代数上のいくつかの定理をE1に追加し（その公理系をE′とする），E′上で，P(gj(...))≡TRUEを示す方が簡単であり，以下の検証ではソートstate についてのみ帰納法を用いる．又，帰納法を用いて検証を行う場合，検証すべき命題”P(s)≅TRUE”だけではその証明は成功しないが，他の命題（補題）”P1(s)≅TRUE”,”P2(s)≅TRUE”,....,”Pk(s)≅TRUE”を加えて，同時併行帰納法で証明が成功することがしばしば起こる．この場合，検証者が必要な補題を適宜追加する必要がある．

(5) プレスブルガーの算術を利用した証明

(2) でも述べたように規定代数の定理を用いて証明を行う場合，すべて検証者が補助定理として導入しなければならず，検証が煩雑になる．しかし，次節以降の検証では，プレスブルガーの算術に関する性質のみを用いて検証できる場合が多く，P(s)≅TRUEの証明に際し，まずP(s)の標準項P(s)↓を求め，次に１．３．４で述べたプレスブルガー文の真偽判定機能を用いることにより，P(s)↓≅TRUEを証明した（定義より，P(s)↓≅TRUEならばP(s)≅TRUEである）．

2．6　検証の具体例

HDLC手順が保持しなければならない種々の性質の内，次に述べる3個の性質を例に，検証の具体的方法や検証に用いた補題等について説明する．

(1) 動作可能性

通信系全体の任意の状態tに対して，tに至るまで規格に従って通信が行われたならば（すなわち，valid(t) $\equiv_E$ TRUE），次に遷移可能な状態が少なくとも1つ存在する．この性質をASL/V言語で記述すると，次のように表される．

［定理1］[valid(s) ⊃ {valid(TsendI.P(s, d, p)) OR ... †
OR valid(Tdrop.B(s))}] $\approx_E$ TRUE

（但し，s，d，pはそれぞれソートstate, data, bool の変数を表す．）

この定理の検証は2．5で述べた場合分けの方法を用いて行った．又，定理が"valid(s) ⊃ P(...)"の形をしているのは，valid の値が"真"の時しか，P(...)の値が定義されていないからである．

(2) P/F ビットが"1"のフレームの送信権

HDLC手順では，P/F ビット"1"のフレームを交互に送信することにより種々の制御を行っているので，1次局，2次局が共にP/F ビット"1"のフレームを送信出来るような状態（ready.P 及び ready.S の値が共に"真"）は存在しない．すなわち，

［定理2］[valid(s) ⊃ {NOT(ready.P(prim(s))) OR
NOT(ready.S(secon(s)))}] $\approx_E$ TRUE

この定理は，以下の定理3などの状態変数間の関係を検証する際の基本的性質である．尚，この定理の検証は次のような意味の補題を導入し，2．5で述べた同時併行帰納法で証明した．

［補題2．1］往路，復路あわせて，伝送路上には，P/F ビット"1"のフレームは高々1個しか存在せず，P/F ビット"1"のフ

---

†　2．3．3で述べた状態遷移関数すべての"OR"を表す．

レームが伝送路上に存在する時，１次局，２次局はP/F ビット”１”のフレームを送信出来ない．

(3) 状態変数間の関係

１，２次局間の状態変数間に成立する不変式の例として次のような定理の検証を行った．

［定理３］ [valid(s) ⊃ order(seq.P(prim(s)), ack.S(secon(s)), low.P(prim(s)))] $\underset{E}{\approx}$ TRUE

(order(x, y, z) は [(x-z) MOD MODULUS] ≧ [(y-z) MOD MODULUS] を略記したものである（表８参照））．

尚，この定理はStenningのプロトコル[7]のようにサイクリックなシーケンス番号を考慮しない場合には，次のように表現される．

seq.P(prim(s)) ≧ ack.S(secon(s)) ≧ low.P(prim(s))

通常，seq.P は送信側で上位レベルから渡されたデータの個数に，ack.S は受信側で正しく受信したデータの個数に，low.P は相手側に受信されたことが送信側で確認されているデータの個数に対応しており，この定理は，３者間の大小関係を表している．この定理の検証には，定理２，補題２．１及び以下に示すような補題（３．１～３．５）を導入し，同時併行帰納法で証明した†．

ack.S の値は受信したＩフレームの送信順序番号とack.S の値が一致した時のみ１増やされる（表12のC032）．このため，ack.S の値が１増えてもseq.P の値を超えないことを示すために，次の補題を導入する．

［補題３．１］往路の伝送路上に送信順序番号がack.S の値に等しいＩフレームが乗っているならば，それ以降（送信側）のＩフレームの送信順序番号は，seq.P の値より小さい．

同様に，low.P の値は受信フレームの受信順序番号にセットされるため（表12のC031），次の補題が必要である．

---

† 以下では，サイクリックなシーケンス番号を考慮しない場合の表現に簡単化されている．

［補題３．２］復路の伝送路上のフレームの受信順序番号はack.S の値より小さくなければならない．

seq.P の値は，P/F ビットが”１”で且つ受信順序番号が checkpoint.P の値より小さいようなフレームを受信した時，その受信順序番号にセットされる（小さい値にもどされる）（表12のC029）．この時，seq.P の値がack.S の値より小さくならないことを示すためには，次の補題が必要である．

［補題３．３］復路の伝送路上にP/F ビットが”１”で且つ受信順序番号がcheckpoint.Pの値より小さいフレームが存在するならば，その受信順序番号は，ack.S の値に等しくなければならない†．

上述の３個の補題が成立すれば，定理３は証明出来るが，これらの補題を証明するのにさらに次のような補題を用いて同時併行帰納法で証明した．

［補題３．４］(イ)復路の伝送路上にP/F ビットが”１”で且つ受信順序番号がcheckpoint.Pの値より小さいフレームが存在する，又は，(ロ)ready.S の値が真で且つack.S の値がcheckpoint.Pの値より小さい，のいずれかが成立する時，往路の伝送路上の任意のＩフレームの送信順序番号はack.S の値に等しくない．

［補題３．５］往路の伝送路上にP/F ビットが”１”のフレームが存在する時，そのフレーム以降のＩフレームの送信順序番号はack.S の値に等しくない．

表16に定理１～３及び補題の証明で用いた基底代数の定理を示す．これらの定理は上述の補題同様，検証の途中で項の書き換えを進めるためにはどのような基底代数の定理が有効かを考えて追加を行った．尚，定理１～３の証明に要した作業日数は約１人月である．

この章の記述に，(1) 送信側で上位レベルから渡されるデータに順にMODULUS を法とする送信順序番号を付けて送信すること，

---

†　補題３．２で受信順序番号はack.S の値より大きくないことが要求されるので，等しい場合しか起らない．

(2) 再送の時のデータは，その送信順序番号を付けて送った直前のデータと同じであること，及び (3) 受信側では正しく受信したデータを順次上位レベルへ渡すことなど，上位レベルとのインターフェイス部分の記述を追加することにより，次のような定理の検証を行うことが出来た[20][21][22]．

［定理４］２次局で上位レベルへ渡されるデータは１次局の上位レベルから渡されたデータと順序も含め同一である．

表16　検証で用いた基底代数の定理

```
EPT1: order(s, s, u) ≅E TRUE ;
EPT2: { order(s, t, u) AND NOT(plus1(s) = u) }
          ⊃ order(plus1(s), t, u) ≅E TRUE ;
EPT3: { order(s, t, u) AND NOT(plus1(s) = u) }
          ⊃ order(plus1(s), plus1(t), u) ≅E TRUE ;
EPT4: NOT(plus1(s) = u) ⊃ order(plus1(s), s, u) ≅E TRUE ;
EPT5: { order(s, t, u) AND order(s, w, t) AND NOT(s = w)
        AND(w = t) } ⊃ order(s, plus1(t), u) ≅E TRUE ;
EPT6: { order(s, t, u) AND order(t, w, u) }
          ⊃ order(s, t, w) ≅E TRUE ;
```

(但し，s, t, u, w は，ソート nn-integer の変数)

２．７　結言

この章では，HDLC手順に従う通信系を抽象的な順序機械と考え，代数的記述を行った．この方法は，状態遷移図等ではコンパクトに扱うことのできないパラメータやシーケンス番号の取り扱いが容易であり，高級言語による方法に比べ，検証が厳密に行えるという特徴を持つ．又，従来の代数的仕様の検証のように項の書き換えのみで検証を行わず，プレスブルガー算術の性質を用いて検証出来る場合は，１．３．４で述べたプレスブルガー文真偽判定機能を用いて機械的に検証を行うことにより，検証が効率的に行うことが出来た．

結　　論

本論文では，まずプレスブルガー文の真偽判定機能を持つ筆者らの検証支援系が，代数的仕様記述の検証に有効であることを示し，次に，HDLC手順のような実際的な規模の検証が，代数的記述の枠組みの中で効率的に行えることを示した．これらの新しい諸成果についてはすでに各章の結言で述べられているので，以下では，今後に残された課題について述べる．

まず，支援系については，プレスブルガー文の真偽判定機能だけでなく，リストや配列に対する性質を機械的に検証する機能を導入すること等が考えられる．又，代数的仕様記述の検証法については，HuetやMusserらによる帰納法を用いない証明法[1]（通常は帰納法を用いなければ証明できない性質を，Knuth-Bendixの完全化アルゴリズムを利用して帰納法を用いずに証明する方法）を本論文で述べたHDLC手順の検証等に適用できるよう拡張すること等が考えられる．

さらに，HDLC手順のようなプロトコルの検証については，本論文で述べたプロトコルの静的な性質だけでなく，"プロトコルが進むかどうか"といった動的な性質の検証が，代数的な枠組みでどの程度可能かを検討すること等が考えられる．これについては，現在文献(8)で検討中である．

謝　　辞

本研究に関して，直接理解ある御指導を賜わり，つねに励ましていただいた嵩忠雄教授に心から深謝します．

大学学部，大学院を通じて御指導いただいた情報工学科の藤沢俊男教授，都倉信樹教授，高島堅助教授，並びに大阪大学産業科学研究所の豊田順一教授に心から感謝します．

本研究の全過程を通じて終始，適切な御指導と御助言をいただいた谷口健一助教授，並びに滋賀大学の森将豪助教授に心から感謝します．

本論文の作成に際し有益な御助言をいただいた電総研の鳥居宏次博士に心から感謝します．

種々の面で御助言，御援助いただいた大阪大学大型計算機センターの藤井護助教授，嵩研究室の杉山裕二助手，伊藤実助手，大阪大学情報処理教育センターの鈴木一郎助手，並びに電電公社横須賀通研の葛山善基博士に心から感謝します．

又，大学学部，大学院を通じて共に研究し励ましあってきた学友の井上克郎氏の有益な御助言，御討論に感謝します．

さらに，筆者の在学中，御討論いただいた嵩研究室の方々，特に第１章で述べた代数的仕様検証支援系の作成に御協力いただいた中西道雄氏（現三菱電機（株）），縄田修造氏（現三菱重工業（株）），田中選氏（現富士通（株）），柳原幸一氏（現日本電気（株）），工藤尊弘氏，樋口昌宏氏に感謝します．

文　　献

(1) 二木，外山：”項書き換え型計算モデルとその応用”，情報処理，24，2，pp.133-146（1983）．

(2) Musser.D.R.：”Abstract data type specification in the AFFIRM system”，IEEE Trans. Softw. Eng.，SE-6，1，pp.24-32（1980）．

(3) 川原林，太田，大山口，稲垣：”代数的仕様記述に基づいたソフトウェアの正当性証明システム”，信学論(D)，J66-D，6，pp.691-698（昭58-06）．

(4) 東野，工藤，杉山，縄田：”代数的仕様の検証支援システムの試作”，信学技報，AL82-27（1982-09）．

(5) 伊理，他：”計算の効率化とその限界”，数学セミナー増刊，入門現代の数学13，pp.97-126（1980）．

(6) 東野，森，谷口，嵩：”代数的に記述されたHDLCプロトコルの検証”，信学論(D)，J66-7，7，pp.773-780（昭58-07）．

(7) Stenning.N.V.：”A data transfer protocol”，Comput. Network，1，2，pp.99-110（1976）．

(8) 工藤，東野，樋口，谷口，嵩，森：”Stenningのプロトコルの代数的記述とその検証”，信学技報，AL83-64（1984-01）．

(9) 杉山，谷口，嵩：”基底代数を前提とする代数的仕様”，信学論(D)，J64-D，4，pp.324-331（昭56-04）．

(10) 電総研：”XXPL言語仕様，使用説明書”（1979）．

(11) 井上，関，杉山，谷口，嵩：”関数的プログラミング言語ASL-Fコンパイラ”，情処全大，pp.13-14，（昭58-03）．

(12) Wirth.N.：”Algorithm + Data Structure = Program”，Prentice-Hall（1976）．

(13) 樋口，柳原：大阪大学基礎工学部卒業論文（1983）．

(14) 斉藤，加藤，猪瀬：”オートマトンモデルによるHDLCプロトコル製品検証の一方式”，信学論(D)，J63-D，8，pp.642-649（昭55-08）．

(15) 森，東野，杉山，谷口，嵩：”HDLC手順の代数的記述”，信学論(D)，J64-D，2，pp.124-131（昭56-02）．

(16) Huet.G. and Oppen.D.C.：”Equations and rewrite relus：A survey”，Formal Language Theory.R.V.Book ed.，Academic Press，pp.349-405（1980）．

(17) ”JIS ハイレベルデータリンク制御手順”，JIS C 6363-6365（昭53-11）．

(18) 電電公社：”DCNAデータリンクレベルプロトコル”，（昭56-04）．

(19) Dershowitz.N. and Manna.Z.：”Proving termination with multiset ordering”，Comp. ACM，22，8，pp.465-476（1979）．

(20) 東野，森，杉山，谷口，嵩：”HDLCの代数的記述と検証”，信学技報，AL80-07（1980-05）．

(21) 東野，縄田，森，谷口，嵩：”代数的記述における検証について－HDLC手順の場合－”，信学技報，AL81-71（1981-11）．

(22) 東野：”HDLC手順の代数的記述と検証”，大阪大学基礎工学部修士論文（昭56-03）．