



Title	通信プロトコルの等価性及びエラーリカバリ性の検証 とプログラムへの変換に関する研究
Author(s)	二宮, 清
Citation	大阪大学, 1990, 博士論文
Version Type	VoR
URL	https://doi.org/10.11501/3052208
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

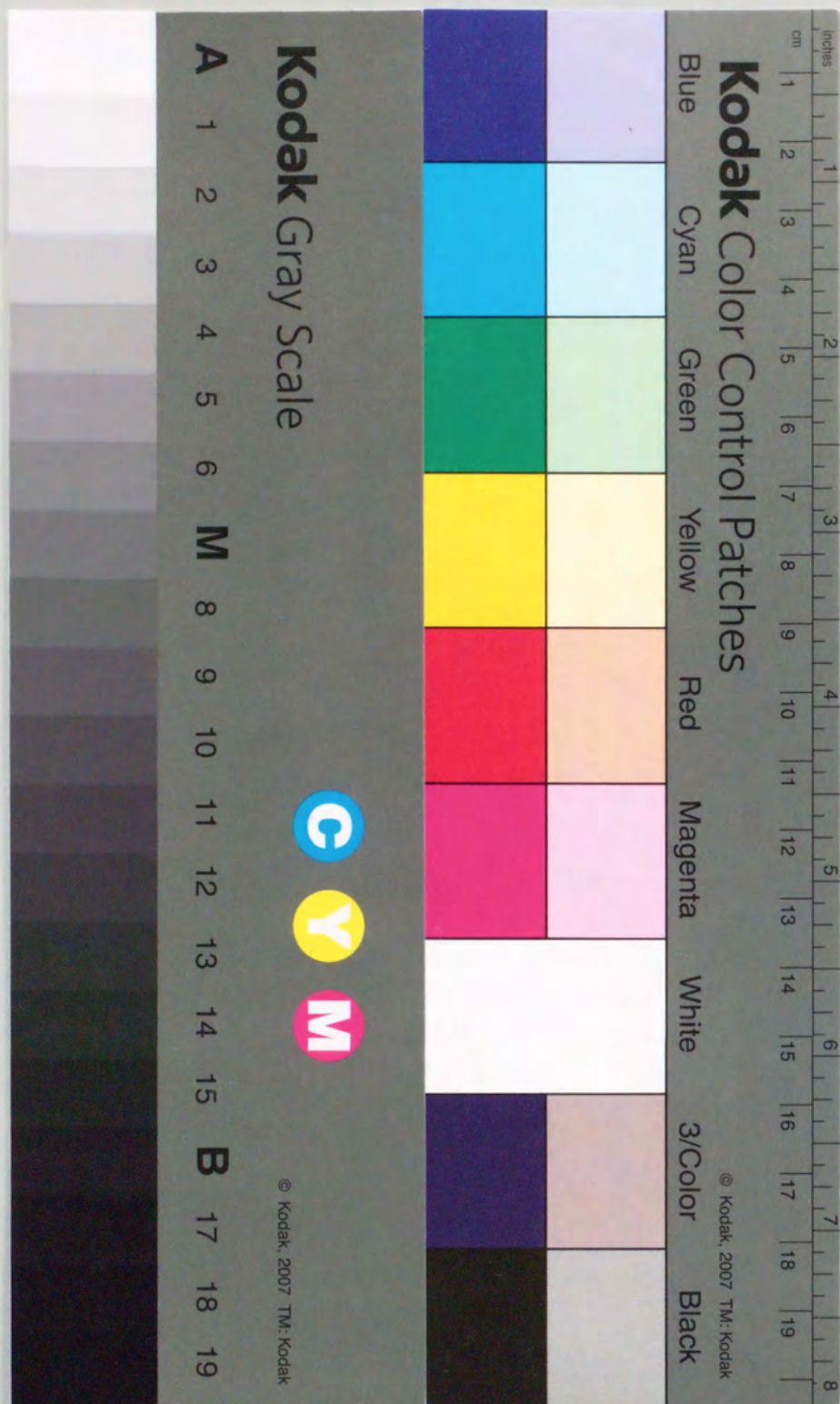
<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

通信プロトコルの等価性及び
エラーリカバリ性の検証と
プログラムへの変換に関する研究

平成 2年 8月

二宮 清



通信プロトコルの等価性及び
エラーリカバリ性の検証と
プログラムへの変換に関する研究

平成 2 年 8 月

二宮 清

内 容 梗 概

本論文は、筆者がダイキン工業株式会社、情報システム部に在職中に通信処理関係の研究の機会を得て、大阪大学基礎工学部情報工学科、嵩研究室と谷口研究室において行った通信処理に関する研究のうち、プロトコルマシンの等価性及びエラーリカバリー性の検証と通信プロトコルの代数的仕様からプログラムへの変換に関する研究を3章にまとめたものである。

緒論及び各章の第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、各章の最後の節及び全体の結論では、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、プロトコルマシンを、任意の整数値を保持する有限個のレジスタを持ち、演算として整数の加減算、等号・不等号判定、AND、OR、NOTを用いるオートマトンとしてモデル化し、その等価性を定義している。一般にこのモデルの等価性は判定不能である。本論文では二つのプロトコルマシンM1, M2の状態やレジスタ値の間関係を表わす述語 Ψ を検証者が指定し、「M1, M2が常に Ψ をみたし、かつ Ψ をみたす状態やレジスタ値に対しては次の入力に対する出力が等しいか」ということを調べる方法を提案する。第1章の前半では、述語 Ψ のクラスを定め、それらが成り立つための十分条件を与え、後半では、その判定手続きをプログラム化して、SDLC手順を実現する二つのマシン例に対して適用し、等価性の証明において本論文の手法が有効であることを確かめた。

第2章では、第1章でモデル化した2つのプロトコルマシンが、エラーリカバリ性を持つことを保証するための自動検証法について述べている。一般に、プロトコルマシン間で通信を行っている際に、マシンダウンや回線障害等が発生して異常な状況に陥ってもいつかは正常な状況に復帰することをエラーリカバリ性と呼ぶが、本論文ではプロトコルマシン対がエラーリカバリ性を持つ事を整数線形計画問題の解の非存在性の証明に帰着して機械的に証明する為の方法を提案した。又、その証明手続きをプログラム化し、それを用いてハイレベル手順を実現するプロトコルマシン対のエラーリカバリ性を検証することによりその手法の有効性を示した。

第3章では、通信プロトコルの代数的仕様から、手続き型プログラムへの変換について述べている。まず、変換法の正しさに関する議論が厳密に行えるよう、変換法を形式的に記述した。また、この変換法に従ってプロトコルの代数的仕様をC言語プログラムに変換するコンパイラを作成した。本コンパイラでは、レジスタ値の待避の個数が少なくなるような代入文の実行順を選択する等の最適化も行っている。OSIセッションプロトコルの代数的仕様を入力例として生成されたプログラムを動作させた結果、人手で作成されたプログラムと同等の効率をもつプログラムを生成できることが確認された。

通信プロトコルの等価性及びエラーリカバリ性の 検証とプログラムへの変換に関する研究

目次

緒論

第1章 通信プロトコルの等価性検証の一方法

- 1.1 序言
- 1.2 プロトコルのモデル化
- 1.3 等価性の定義
- 1.4 等価性の検証法
- 1.5 検証例
- 1.6 結言

第2章 通信プロトコルにおけるエラーリカバリ性 検証の一方法

- 2.1 序言
- 2.2 プロトコルのモデル化
- 2.3 通信系の動作
- 2.4 エラーリカバリ性の定義
- 2.5 エラーリカバリ性の検証法
- 2.6 検証例
- 2.7 パラメータ値に依存しない検証法
- 2.8 結言

第3章 通信プロトコルの代数的仕様からプログラム への変換

- 3.1 序言
- 3.2 抽象的順序機械
- 3.3 プログラムへの変換法
- 3.4 コンパイラの実現と変換例
- 3.5 結言

結論

謝辞

文献

緒論

近年のデータ通信の発展に伴ない、情報の送受信の
手順を規定する通信プロトコル（以下単にプロトコル
と呼ぶ）の厳密かつ形式的な記述やその正しさの検証、
さらには、プロトコルに従って動作する通信プログラ
ムの開発を容易にし、その開発コストを減少させる為
のプロトコル記述法やプログラム設計法等が重要な課
題になってきており、様々な研究が行われている。⁽¹⁾

一般に、プロトコルは選択性のあるいくつかの機能
群から構成されている場合が多く、実際のプロトコル
マシン（または単にマシンと呼ぶ）の開発に際しては、
それぞれのマシンに必要な機能のみがインプリメント
される。又、タイマの設定値や再送回数の上限等はパ
ラメータ化されていることが多く、各マシンの設計者
がマシンの開発時に具体的にその値を設定する。この
ため、同じプロトコルに基づいて開発されたマシンの
組であっても、等価性（同一の入力系列（マシンの遷
移要因の系列）に対して、出力系列が同じであること）
が保証されないことが多い。このため、一つのマシン
を別のマシンに更新する（古いマシンを新しいマシン
に置換えたり、あるメーカーのマシンを、別のメーカ
ーのマシンに置換える）と、それまで通信できた相手
と通信できなくなる場合がある。マシンの等価性が保
証されれば更新が可能となる。しかし、従来、マシン
の等価性を証明する有効な方法がなく、テスト入力に
よる検査（幾つかのテスト系列を入力して、同一の出
力系列が得られるかどうかをチェックする）が行われ
てきたが⁽¹⁾、このようなテストでは必ずしも等価性が

保証されない、又、検査にはかなりの労力を要する。

第1章では2つのプロトコルマシンの等価性証明の一方法について述べる。まず、各マシンを、「有限制御部」と任意の非負整数値を保もする有限個の「レジスタ」を持ち、演算として整数の加減算、等号・不等号判定、AND, OR, NOTを用いるオートマトンとしてモデル化し、その等価性を定義している。一般的にデータリンク層のプロトコルマシンの多くが、本論文で提案するオートマトンとしてモデル化できる。また、このモデルを用いていわゆる2-計数機械の動作を模倣することができる。2-計数機械はチューリング機械と原理的な能力が同じであること、及びチューリング機械の等価性が判定不能であることが知られているので、このモデルにおける等価性も一般に判定不能である。このため、2つのマシンM1, M2の状態やレジスタ値の間で動作途中において常に成り立つと思われる関係 Ψ を検証者が指定し、

(1) 任意の入力系列に対してM1とM2が到達する状態とその時のレジスタ値の組が関係 Ψ を満足すること、及び(2) 関係 Ψ を満たすどのような状態とレジスタ値についても、任意の一つの入力に対してM1とM2の出力が同一であることを、計算機を用いて証明するという方法を採用する。本論文では、 Ψ のクラスをプレスブルガー文の部分クラスに制限し、M1, M2, Ψ が(1), (2)を満たすための一つの十分条件を導き出した。その十分条件が成り立つか否かの判定問題を整数線形計画問題の解の非存在性の証明に帰着して機械的に証明するための方法について述べる。又、十分条件の判定手続きをプログラム化して、SDLC手順の2次局に相当する2つのマシンの等価性を証明した結果についても述べる。

さて、一般にプロトコルは、正常な状況（正常に送受信を行っているような状況）における各プロトコルマシン（通信制御装置）の動作を記述した部分と、異常な状況（回線に障害等が発生したような状況）からどのように正常な状況に復帰するかを記述した部分、に分類されることが多い。この際、「異常な状況からいつかは正常な状況に復帰すること」（エラーリカバリ性をもつという）を機械的に証明できれば望ましい。しかし、従来、エラーリカバリ性を証明する有効な方法がなく、テスト手法による検査が行われてきたが、このようなテスト手法では必ずしもエラーリカバリ性を論理的に保証できない。

第2章では、第1章でモデル化した2つのプロトコルマシンがエラーリカバリ性をもつことを保証するための自動検証法について述べる。第1章と同様に各プロトコルマシンは、「有限制御部」と任意の整数値を保持する有限個の「レジスタ」を持ち、演算として整数の加減、等号・不等号判定、AND, OR, NOTを用いるオートマトンとして記述する。このモデルで記述された2つのプロトコルマシンM1, M2の状態とレジスタ値の組（以下、M1, M2対の全状態、または単に全状態と呼ぶ）を引数とする述語 Φ を線形不等式の論理結合の形で指定する。述語 Φ の値は、その値が真の時は「正常な状況」を表すように指定し、「M1, M2の初期状態と初期レジスタ値の組から、M1とM2が（通信回線の誤り等も考慮して）送受信等の動作を行うことによって遷移する任意の全状態に対して、たとえその全状態が Φ を満足しない全状態であっても、それ以降可能などのような動作を続けてもいつかは再び Φ を満足する全状態に遷移する」時、M1, M2は Φ に対するエ

ラーリカバリ性をもつと定義する。

提案するモデルのマシン及び Φ のクラスにおいては、M1、M2が Φ に対するエラーリカバリ性をもつ場合でも、そのことを証明するアルゴリズムが存在しない。そこで、必ずしも証明に成功する保証はないが、「 Φ を満足する全状態からどのような動作を行っても再び Φ を満足する全状態へ遷移すること」を保証するための一つの手続きを与え、その手続きを再帰的に用いることによって、M1、M2が Φ に対するエラーリカバリ性をもつことを証明する。また、検証の自動化のため、与えられたプロトコルマシン対が与えられた Φ に対するエラーリカバリ性をもつことを、第1章と同様に整数線形計画問題の解の非存在性の証明に帰着して機械的に証明するための方法を提案する。尚、一般にこのような検証法では、HDLCにおけるウィンドサイズや最大再送回数等のようなプロトコルの「パラメータ値」が増大すると、検証に要する時間が急激に増大する。2章の後半では、本検証法を改良して、このようなパラメータ値に依存しない計算時間で検証を行う手法を述べる。また、提案する証明手続きを用いてHDLC手順に従うあるプロトコルマシン対のエラーリカバリ性を検証することにより、本手法の有効性を示す。

さて、与えられたプロトコル仕様から、それに従って動作するプログラムを導出する手法を検討することは、通信ソフトウェアの設計・開発の際の最も基本的で重要な課題である。与えられたプロトコルを通信プログラムに機械的に変換できるためには、プロトコル自身が、形式的意味の定義された言語で記述されていることが必要である。ISOではプロトコル仕様の形

式的記述言語として、LOTOS, Estelleを採用しており⁽³⁰⁾⁽³¹⁾、これらの言語で記述されたプロトコル仕様の解釈実行法や、通信プログラムへの変換法に関する研究がなされている⁽³²⁾⁽³³⁾⁽³⁴⁾⁽³⁵⁾。LOTOSでは、仕様の厳密な意味は、通信系の取り得る動作系列の集合として定義されている。そのため、OSI規格のように、状態遷移モデルに基づいて記述された仕様に対応するレベルで、自然に記述するのは困難である。同様に、LOTOS仕様と順序機械的に動作する通信プログラムとの間には、記述レベルにかなりの隔たりがあり、プログラムへの変換法の正しさを厳密に証明するのは容易ではないと思われる。一方、Estelleでは、拡張状態遷移モデルに基づいて仕様の意味が定義されているので、比較的容易に効率の良いプログラムに変換できるという利点をもつ。反面、上位層に提供する通信サービスを抽象的なレベルで記述することが困難である。

第3章では、抽象的順序機械と呼ばれる形で記述された通信プロトコルの代数的仕様から、手続き型プログラムへの変換法について述べる。代数的仕様記述法は、厳密な意味の定義が簡明であり、検証の際に必要な仕様の無矛盾性や詳細化等の概念も、合同関係という概念のみを用いて簡明に定義できる。代数的仕様の部分クラスとして、抽象的順序機械の形で記述された仕様⁽¹⁸⁾⁽¹⁹⁾がある。この仕様では、論理型、整数型等の基本タイプを前提として、状態の構造を陽にはもたない抽象的状态を表すタイプを定義する。関数には状態遷移関数、状態成分関数（抽象的状态からその成分の値を取り出す関数）、および補助関数があり、各状態遷移後の状態成分関数の値を公理によって定義する。抽象的状态は構造を陽にはもたないことから、新

たに状態成分を追加する際、既に記述した部分を変更することなく、状態成分関数に関する構文と公理を局所的に追加するだけでよい、という特長がある。また、既にHDLCやOSIセッション層等の通信プロトコルを抽象的順序機械の形で記述し、仕様の形式的検証が行われ、本記述法の有効性が実証されている。

仕様からプログラムへの変換法の正しさを厳密に議論するためには、変換法を形式的に記述しておくことが望ましい。第3章前半では、目的プログラムを解釈実行する機械を定義する代数的仕様、および、変換法自身を定義する代数的仕様を記述することにより、変換法を形式的に定義している。また、第3章後半では、この変換法に従って通信プロトコルの代数的仕様をC言語プログラムに変換するコンパイラについて述べる。本コンパイラでは、レジスタ値の待避の個数が少なくなるような代入文の実行順を選択する等の最適化も行っている。OSIセッションプロトコルの代数的仕様を本コンパイラを用いてプログラムに変換し、いくつかのテスト系列についてもそのプログラムを動作させた結果について報告されている⁽²⁹⁾。そこでは、人手で作成したプログラムと同程度の効率をもつプログラムが生成できることが確認されている。

(関連発表論文)

学術論文誌

- (1) 二宮清, 東野輝夫, 谷口健一, 木本智久: "プロトコルマシンの等価性証明の一方法", 電子情報通信学会論文誌(D), Vol. J71-D, No. 12, pp. 2630-2639 (昭63-12).
- (2) 遠一光, 栗屋英司, 関浩之, 藤井護, 二宮清: "抽象的順序機械の形で記述された代数的仕様からプログラムへの変換について", 電子情報通信学会論文誌(D-I), Vol. J73-D-I, No. 2, 1-3, 5, 6節, pp. 201-206, 208-210 (平2-02).
- (3) 木本智久, 二宮清, 東野輝夫, 谷口健一, 森将豪: "通信プロトコルにおけるエラーリカバリ性の自動検証の一方式", 電子情報通信学会論文誌(D-I), Vol. J73-D-1, No. 5, pp. 500-509 (平2-05).

国際会議発表

- (1) T. Higashino, K. Ninomiya, T. Kimoto, K. Taniguchi and M. Mori: "Automated Verification of Equivalence of Protocol Machines", Protocol Specification, Testing, and Verification, IX, pp. 235-246, North-Holland (1990).

プロトコルマシンの 等価性検証の一方法

1.1 序言

一般に、プロトコル（通信規約）は選択性のあるいくつかの機能群から構成されている場合が多く、実際のプロトコルマシン（または単にマシンと呼ぶ）の開発に際しては、それぞれのマシンに必要な機能のみがインプリメントされる。又、タイマの設定値や再送回数の上限等はパラメータ化されていることが多く、各マシンの設計者がマシンの開発時に具体的にその値を設定する。このため、同じプロトコルに基づいて開発されたマシンの組であっても、等価性（同一の入力系列（マシンの遷移要因の系列）に対して、出力系列が同じであること）が保証されないことが多い。このため、一つのマシンを別のマシンに更新する（古いマシンを新しいマシンに更新したり、あるメーカーのマシンを、別のメーカーのマシンに更新する）と、それまで通信できた相手と通信できなくなる場合がある。マシンの等価性が保証されれば更新が可能となる。しかし、従来、マシンの等価性を証明する有効な方法がなく、テスト入力による検査（幾つかのテスト系列を入力して、同一の出力系列が得られるかどうかをチェックする）が行われてきたが⁽¹⁾、このようなテストでは必ずしも等価性が保証されない。又、検査にはかなりの労力を要する。

第1章では、マシンの等価性を証明するための一つの方法を提案する。まず、筆者らは、プロトコルマシンを、整数値を保持する有限個のレジスタ（送受信のためのデータやシーケンス番号等は整数化して取り扱う）を持つオートマトンとしてモデル化した。1.2ではそのモデルについて述べ、1.3ではこのモデルにお

けるマシンの等価性を定義している。このモデルは、特定の階層のプロトコルマシンを対象としたものではないが、例えば、データリンク層⁽¹⁾のプロトコルマシンの多くが第1章で提案するオートマトンとしてモデル化できる。

このモデルを用いて、いわゆる2-計数機械（two-counter machine）の動作を模倣することができる。2-計数機械はチューリング機械と等価であること、及び、チューリング機械の等価性が判定不能であることが知られているので⁽²⁾、上述のモデルにおける等価性も一般には判定不能である。そこで、等価であるための次のような十分条件を考える。今、 Ψ を2つのマシンM1, M2の状態やレジスタ値の間の関係を表す述語とする。この時、次の(1), (2)が共に成り立てば、M1とM2は等価である。

- (1) 任意の入力系列に対して、M1とM2が到達する状態とその時のレジスタ値の組が関係 Ψ を満足する。
- (2) 関係 Ψ を満たすどのような状態とレジスタ値についても、任意の一つの入力に対してM1とM2の出力が同一である。

第1章では、検証者（等価性の証明を行う者）が関係 Ψ を指定（予測）し、計算機を用いて(1), (2)を証明することによって等価性を証明する方法を採用する。まず、関係 Ψ は1.2で述べるP文（状態とレジスタ値を表す変数の線形不等式、またはそれらの不等式をand, or, notで結合した論理式、存在作用素 $\exists$ を持たないプレスブルガー文に相当）で指定されるものとし、1.4ではM1, M2及びP文 Ψ が(1), (2)を満たすための一つの十分条件を与え、M1, M2, Ψ がその十分条件を満足する時、且つその時のみ、M1とM2は Ψ 等価であると定義す

る。定義より $M1$ と $M2$ が Ψ 等価なら $M1$ と $M2$ は等価である。次に、 $M1$ と $M2$ が Ψ 等価であるか否かの判定問題を整数線形計画問題の解の非存在性の判定問題に帰着して判定する方法について述べる。1.5では実際にワークステーション (IBM RTPC) 上で作成した Ψ 等価性の判定手続き⁽³⁾を用いてSDLC手順の2次局⁽⁸⁾に相当する2つのマシンの等価性を証明した結果について述べる⁽⁴⁾。

提案する方法を用いて等価性を証明する場合、検証者自身が関係 Ψ を P 文で指定 (予測) しなければならない。しかし、 Ψ を指定すれば Ψ 等価であるか否かは Ψ 等価性の判定手続きを用いて機械的に判定できるという特徴を持つ。尚、提案する方法は等価性を証明するための一つの十分条件であり、一般には2つのマシン $M1, M2$ が等価であっても Ψ 等価となるような P 文 Ψ が必ず存在するとは限らないし、そのような Ψ が存在してもその Ψ を検証者が指定できるとは限らない。また、 Ψ 等価性の判定手続きは多項式時間算法ではない。しかし、筆者らが記述した幾つかのマシンの組に対しては上述の方法で等価性の証明が実際に行えたので、第1章で提案する方法は実用のマシンの等価性の証明に幅広く適用出来ると考えられる。

1.2 対象とするプロトコルマシンのモデル化

1.2.1 プレスブルガー算術

第1章では、プロトコルマシンのモデル化や等価性の判定にプレスブルガー算術⁽⁵⁾を用いる。以下、第1章で用いる用語や記法について簡単に述べる。

N を整数の集合とし、 N の変数 $(x, y, z, \dots)$ 、 N の定数 $(0, \pm 1, \pm 2, \dots)$ 及び加減算の演算子 $(+, -)$ から成

る項 (N の変数の1次結合 (線形結合)⁺) を「 P 項」と呼ぶ。又、「 P 文」を次のように定義する。

- (1) t_1, t_2 が P 項の時、 $t_1 = t_2, t_1 < t_2, t_1 \leq t_2, t_1 \geq t_2, t_1 > t_2$ は P 文である。
- (2) A, B が P 文ならば、 $(A) \text{ or } (B), (A) \text{ and } (B), (A) \supset (B), \text{ not } (A)$ は各々 P 文である ($\supset$ は、含意を表す)。
- (3) 上の (1), (2) を有限回適用して得られるもののみが P 文である。

変数を含まない P 文に対しては、演算子 $(+, -, =, <, \leq, \geq, >, \text{ or }, \text{ and }, \supset, \text{ not })$ の通常の意味に従って、その真偽を定める (例えば、 P 文「 $(3+5 \leq 10) \text{ or } (7=2)$ 」の値は真)。

さらに、 P 文 A に含まれる全ての変数 $x_1, \dots, x_n$ を全称記号 $\forall$ で束縛した論理式を $\forall x_1, \dots, x_n (A)$ で表す。論理式 $\forall x_1, \dots, x_n (A)$ において、 P 文 A の変数 $x_1, \dots, x_n$ にどのような整数を代入しても、代入した各 P 文が真である時、且つその時のみ、論理式 $\forall x_1, \dots, x_n (A)$ は真であるという。尚、論理式 $\forall x_1, \dots, x_n (A)$ の真偽は、整数線形計画問題の解の非存在性の判定手続きを用いて判定できる⁽⁶⁾。

1.2.2 プロトコルマシンのモデル化

第1章では、プロトコルマシンを、「有限制御部」及び有限個の整数値を保持する「レジスタ」から成るオートマトンとしてモデル化する。「相手局からのフレームの受信」や「タイマーからの割り込み」、「相手局へのデータの送信」、あるいは、「相手局へのデータの再送」⁺⁺などを、各々、このオートマトンの「入力記号」に対応付ける。入力記号の種類は有限個とするが、各入力記号は有限個の整数値を「パラメータ」

として持ってもよいとする（以下では、入力記号とパラメータ値の組を「入力」と呼ぶ）。例えば、相手局からフレームを受信する場合、フレームの受信という事象が入力記号であり、そのフレームに含まれる受信データやシーケンス番号がパラメータ値に相当する。また、相手局への送信フレームや上位レベルへの受信通知、タイマーへのセット／リセット信号等をこのオートマトンからの「出力」に対応付ける。出力は、整数の d 字組とし（ d はマシンで決まる定数）、 d 字組の各要素は、通信相手局、上位レベル、タイマー等へ受け渡される内容である（入力パラメータ値同様、出力も整数化して取り扱う）。

オートマトンは、現在の状態（以下、状態は有限制御部の状態を表す）、レジスタ値及び入力（入力記号とパラメータ値の組）の組に対して、次の状態、次のレジスタ値及び出力が定義されるいわゆる Mealy 型のオートマトンとして定義する。また、すべての状態、レジスタ値、入力の組に対して、次の状態、次のレジスタ値、出力が「完全」に指定される必要はなく、特定の状態、レジスタ値、入力の組に対して、次の状態、次のレジスタ値、出力が指定されないいわゆる「不完全」に指定されたオートマトンでもよい。以下、プロトコルマシン M を形式的に次の 7 字組として定義する。

+ 変数を含まない特別な場合も含む。

++ 第 1 章ではマシンの状態を変化させるような各要因を、各々「入力記号」と見なしている。従って、送信や再送のようなマシン内部からの自発的な動作も「入力記号」とみなす。

[プロトコルマシン $M = \langle S, R, S_{R1}, I, \sigma, \delta, \rho \rangle$]

$S = \{s_1, \dots, s_k\}$: 有限個の状態集合。

但し、各状態 $s_1, \dots, s_k$ に適当な非負整数を割り当てる。

$R = \langle R_1, \dots, R_n \rangle$: レジスタ名の n 字組（ n はマシン M のレジスタの数）。

各レジスタ R_i は整数値を保持する。

$S_{R1} = \langle s_1, r_{11}, \dots, r_{1n} \rangle \in S \times N^n$:

マシン M の初期状態 s_1 とレジスタ $R_1, \dots, R_n$ の初期値（整数値） $r_{11}, \dots, r_{1n}$ の $n+1$ 字組。

$I = \{\alpha_1, \dots, \alpha_m\}$: 有限個の入力記号の集合。

各入力記号 α_i は（ α_i で決まる）有限個のパラメータを保持してもよい。パラメータ値は整数とし、入力記号 α_i のパラメータ値が

$q_{i1}, \dots, q_{ij}$ の時、入力を $\alpha_i(q_{i1}, \dots, q_{ij})$ で表す。また、 α_i のパラメータ数を $|\alpha_i|$ で表す。

$I_p = \{\alpha_i(q_{i1}, \dots, q_{ij}) \mid$

$\alpha_i \in I \text{ 且つ } q_{i1}, \dots, q_{ij} \in N \text{ 且つ } |\alpha_i| = j\}$

とおく。 I_p は入力（入力記号とパラメータ値の組）の集合である。

$\sigma : S \times N^n \times I_p \rightarrow S$, 次の状態を定める関数。

現在の状態 s 及びレジスタ値 $r_1, \dots, r_n$ 及び入力 $\alpha_i(q_{i1}, \dots, q_{ij})$ から次の状態を定める。

$\delta : S \times N^n \times I_p \rightarrow N^n$, 次のレジスタ値を定める関数。

現在の状態 s 及びレジスタ値 $r_1, \dots, r_n$ 及び入力 $\alpha_i(q_{i1}, \dots, q_{ij})$ からレジスタ $R_1, \dots, R_n$ の次の値（次のレジスタ値）の n 字組を定める。

$\rho : S \times N^n \times I_p \rightarrow N^d$, 出力を定める関数（ d はマシン M の出力先の数）。

現在の状態 s 及びレジスタ値 $r_1, \dots, r_n$ 及び入力

$\alpha_1(q_{11}, \dots, q_{1j})$ から出力 (整数の d 字組) を定める。 ■

次に, σ, δ, ρ の記述法, ならびに第 1 章で対象とするプロトコルマシンのクラスについて, 図 1.1 の例を用いて説明する。

[例 1 プロトコルマシン M1]

図 1.1 のプロトコルマシン M1 は, BSC 手順⁽⁷⁾の送信局を単純化したものである。M1 の 4 つの状態 s_1, s_2, s_3, s_4 は, 各々初期状態 (ack 受信状態), 送信状態 (ack/nak 待ち状態), 再送準備状態 (nak 受信状態), 最終状態を表しており, R1 は送信回数を, R2 は送信データを保持するレジスタである (何れも整数値を保持する)。データ d の送信を T(d) で, 再送を Re で, 相手局からの受信確認を A(a) で表す (d は実際の送信データを整数化したもの。又, a が 1 の時 ack を, 0 の時 nak を受信したとみなす)。出力先は相手局 1 か所のみとする。M1 は初期状態から送信を開始し, 各送信データに対して ack を受信すれば次のデータを送信し, nak を受信すれば元のデータを再送する。10 個の送信データすべてに対して ack の受信確認を行えば動作を終了する。 ■

図 1.1 において, グラフの頂点は, マシンの状態に対応し, グラフの有向辺によって, 各入力に対する次の状態, レジスタ値, 出力を指定する。グラフの各辺には, 4 字組のラベル <①入力記号, ②条件式, ③次のレジスタ値, ④出力> が付加されている。各状態 (頂点) で①の入力記号が与えられ, その時点のレジスタ値や入力パラメータ値が②の条件式を満足する時, その辺の指す状態 (頂点) に遷移し, ③の指定に従っ

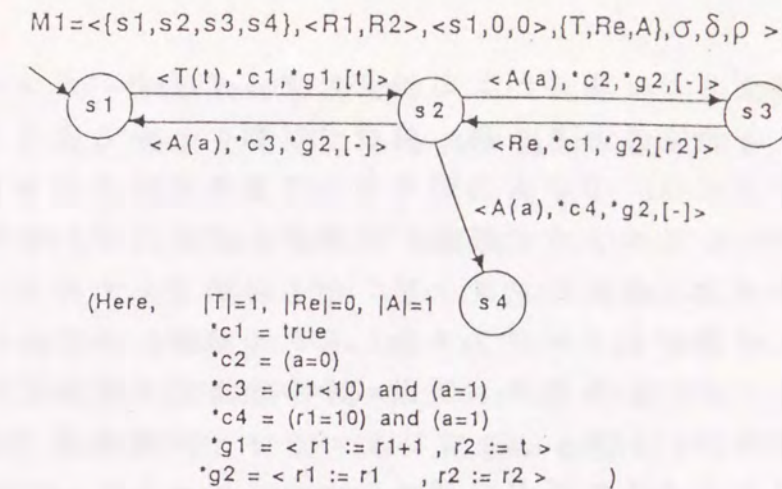


図 1.1 プロトコルマシン M1

てレジスタ値が更新され, ④の指定に従って出力を出す。与えられた入力記号を持つ辺が存在しなかったり, 存在する場合でも, その時点のレジスタ値と入力パラメータ値が②の条件式を満足するような辺が存在しない場合, 次の状態, レジスタ値, 出力は指定されない。尚, ①の入力記号が同じで②の条件式が異なるような辺が一つの頂点から複数個出てもよいが, どのようなレジスタ値, 入力パラメータ値に対しても, 2 個以上の条件式が共に真にならないように各条件式を指定しなければならない。

例えば, 現在の状態が s_1 で, レジスタ R1, R2 の値が各々 n_1, n_2 で, 入力が T(d) の場合, 辺 $s_1 \rightarrow s_2$ に付加されたラベルの①の入力記号が T(t) で (t は T の入力パラメータ値を表す変数), ②の条件式 *c1 が恒真 (true) であるので, 任意のパラメータ値 d に対して次の状態は s_2 となる。又, この遷移において, レジスタ R1, R2 の値は, ③の *g1 の指定に従って, 各々 $n_1+1,$

d に変更され、④の[t]の指定に従って入力パラメータ値dが出力される。尚、頂点(状態) s_1 から出る有向辺の中には、①の入力記号がReであるような辺が存在しない。このような場合、状態が s_1 で入力Reの時の次の状態、次のレジスタ値、出力は指定されない。

又、状態が s_2 でレジスタR1, R2の値が、各々 n_1, n_2 で入力A(k)の場合、頂点 s_2 から出る辺のうち①の入力記号がA(a)(aはAの入力パラメータ値を表す変数)であるような辺が3個($s_2 \rightarrow s_3, s_2 \rightarrow s_1, s_2 \rightarrow s_4$)が存在する。この場合、②の条件式*c2, *c3, *c4の変数 r_1, r_2, a に値 n_1, n_2, k を代入し、代入した論理式が真となるような辺の指す状態(s_3, s_1, s_4 の何れか)に遷移する(どの条件式も満足しなければ、次の状態、レジスタ値、出力は指定されない)。

尚、辺 $s_2 \rightarrow s_3$ のように④の出力として[-]が指定されている場合、何も出力しないことを表し、辺 $s_3 \rightarrow s_2$ のように④の出力として[r2]が指定されている場合、その時点のレジスタR2の値を出力することを表す■

以下では、図1.1のグラフをM1の仕様と呼ぶ。図1.1のような仕様記述法(σ, δ, ρ の指定法)は、状態遷移図と処理状態遷移図を用いた通常の仕様記述法に対応している。尚、第1章では仕様記述に際して、次のような制約を設けている(これらの制約は、1.4で等価性の十分条件を与える際に利用する)。

- (1) 各頂点(状態)から出る辺の数は有限である。
- (2) ②の条件式は、マシンの現在のレジスタ値を表す変数の組 X_n と、入力記号 α_i のパラメータ値を表す変数の組 P_{ij} を変数とするP文で指定する。
- (3) ③の次のレジスタ値及び④の出力は、現在のレジスタ値と入力パラメータ値を表す変数(X_n, P_{ij})

の一次結合で指定する。又、何も出力しない場合も、便宜上そのことを表す適当な整数が出力され则认为する。■

次に「定義域を指定する関数」について述べる。図1.1では、頂点 s_2 から①の入力記号がA(a)であるような辺が3個($s_2 \rightarrow s_3, s_2 \rightarrow s_1, s_2 \rightarrow s_4$)出ている。これらの辺の②の条件式*c2, *c3, *c4の論理和を、状態 s_2 、入力記号Aに対する「定義域を指定する関数」と呼ぶ。但し、入力記号Reのように、状態 s_2 から①の入力記号がReであるような辺がない場合、Reに対する定義域を指定する関数は恒偽(false)と定義する。以下では、状態s、入力記号 α_i に対する定義域を指定する関数を

$$\eta(s, \alpha_i)(X_n, P_{ij})$$

で表す。上述の制約(1),(2)より、定義域を指定する関数はP文となる。

1.3 等価性の定義

等価性を定義するため、1.2.2で述べた関数 σ, δ, ρ の定義を拡張し、次のような関数 $\hat{\sigma}, \hat{\delta}, \hat{\rho}$ を定める。
[$\hat{\sigma}, \hat{\delta}, \hat{\rho}$ の定義]

- (1) $\hat{\sigma}: (Ip)^* \rightarrow S$, 初期状態と初期レジスタ値に有限長の入力系列wを与えた時に到達する状態を指定する関数で、次のように定義する。

④初期状態の指定

$$\hat{\sigma}(\varepsilon) \triangleq s_1 (\varepsilon: \text{空入力系列})$$

⑧ 各入力 $\alpha_i(v_1, \dots, v_j)$ に対する次の状態の指定

$$\hat{\sigma}(w \cdot \alpha_i(v_1, \dots, v_j))$$

$$\triangleq \sigma(\hat{\sigma}(w), \hat{\delta}(w), \alpha_i(v_1, \dots, v_j))$$

(但し, w は任意の入力系列を, $v_1, \dots, v_j$ は,

α_i の任意の入力パラメータ値を表す) ■

(2) $\hat{\delta}: (Ip)^* \rightarrow N^n$, 初期状態と初期レジスタ値に有限長の入力系列 w を与えた時に定まるレジスタ $R_1, \dots, R_n$ の値の n 字組を指定する関数.

(3) $\hat{\rho}: (Ip)^* \rightarrow (N^d)^*$, 初期状態と初期レジスタ値に有限長の入力系列 w を与えた時の出力系列を指定する関数. ■

さらに, 2.2 で定めた定義域を指定する関数 $\eta \langle s, \alpha_i \rangle$ の定義を拡張し, 次のような補助関数 $\hat{\eta}$ を導入する.

[$\hat{\eta}$ の定義]

$\hat{\eta}: (Ip)^* \rightarrow \text{bool}$, 初期状態と初期レジスタ値に有限長の入力系列 w を与えた時に, 出力系列 $\hat{\rho}(w)$ が指定されるか否かを定める関数. すなわち,

$$\hat{\eta}(\varepsilon) \triangleq \text{true}$$

$$\hat{\eta}(w \cdot \alpha_i(v_1, \dots, v_j))$$

$$\triangleq \hat{\eta}(w) \text{ and }$$

$$\eta \langle \hat{\sigma}(w), \alpha_i \rangle \langle \hat{\delta}(w), \langle v_1, \dots, v_j \rangle \rangle$$
 ■

以下, 整数の d 字組 $b \triangleq \langle b_1, \dots, b_d \rangle$, $c \triangleq \langle c_1, \dots, c_d \rangle$ に対して, $b_1 = c_1$ 且つ $b_2 = c_2$... 且つ $b_d = c_d$ が成り立つ時, 且つその時のみ $b = c$ と定義する. また, $x (\triangleq b \cdot x')$, $y (\triangleq c \cdot y')$, $x', y' \in (N^d)^*$ に対して, $b = c$ 且つ $x' = y'$ である時, 且つその時のみ $x = y$ と定義し, マシンの等価性を次のように定義する.

[プロトコルマシンの等価性]

入力記号の集合 I が同じで, 出力が共に整数の d 字組であるような2つのマシン

$$M1 = \langle S, R, S_{R1}, I, \sigma, \delta, \rho \rangle$$

$$M2 = \langle S', R', S_{R1}', I, \sigma', \delta', \rho' \rangle$$

に同一の入力系列 w を与えた時, w に対する出力系列が共に指定され同一, あるいは, 共に指定されない時 (すなわち,

$$\forall w \in (Ip)^*$$

$$[\{ \hat{\eta}(w) \text{ and } \hat{\eta}'(w) \text{ and } \hat{\rho}(w) = \hat{\rho}'(w) \}$$

$$\text{or } \{ \text{not}(\hat{\eta}(w)) \text{ and } \text{not}(\hat{\eta}'(w)) \}]$$

が成り立つ時), 且つその時のみ, $M1$ と $M2$ は等価であるという (但し, $\hat{\eta}$, $\hat{\eta}'$ は, 各々 $M1$, $M2$ の出力系列が指定されているか否かを表す関数). ■

第1章では, 互換性の立場より等価性を上述のように定義している. しかし, 等価性を「 w に対する出力系列が共に指定され同一, あるいは, 少なくとも片方のマシンの出力系列が指定されない」と定義した場合等でも, 以下の議論を若干変更することによって, それぞれの等価性を議論できる.

1.4 等価性の検証法

一般に, 1.2.2 で定義した2つのマシンが等価であるか否かは決定不能である⁽²⁾. そこで, 第1章では検証者が2つのマシン $M1, M2$ の状態とレジスタ値の間で動作途中において常に成り立つと思われる関係 Ψ を P 文で指定し, $M1, M2$ 及び P 文 Ψ が序言で述べた2つの条件(1), (2)を満足することを計算機を用いて証明するという方法を採用する. 今, u, v を各々 $M1$, $M2$ の

状態を表す変数とし (1.2.2 の定義より, 状態を整数に対応付けているので, u, v は整数値を取る変数), B_n, C_h を各々2つのマシン M_1, M_2 のレジスタ値を表す変数の組とする. 又, $\Psi(u, v, B_n, C_h)$ を u, v, B_n, C_h を変数とする P 文とする. 序言で述べた2つの条件 (1), (2) に相当することを 1.2.2 及び 1.3 で述べた表記法に従って記述すると, 次のようになる.

M_1, M_2 及び P 文 Ψ が次の (1), (2) を満足すれば, M_1, M_2 は等価であることは容易に分かる.

(1) $\forall w \in (I_p)^*$

[$\{\hat{\eta}(w) \text{ and } \hat{\eta}'(w) \text{ and } \Psi(\hat{\sigma}(w), \hat{\sigma}'(w), \hat{\rho}(w), \hat{\rho}'(w))\}$
or $\{\text{not}(\hat{\eta}(w)) \text{ and } \text{not}(\hat{\eta}'(w))\}$]

(2) $\hat{\eta}(w), \hat{\eta}'(w)$ の値が真で $s = \hat{\sigma}(w), s' = \hat{\sigma}'(w)$, であるような入力系列 w が存在する任意の状態の組 s, s' に対して,

$\forall B_n, C_h, \alpha_i, P_{ij}$
[$\Psi(s, s', B_n, C_h) \supset$
 $\{\eta(s, \alpha_i)(B_n, P_{ij}) \text{ and } \eta(s', \alpha_i)(C_h, P_{ij}) \text{ and } \rho(s, B_n, \alpha_i(P_{ij})) = \rho(s', C_h, \alpha_i(P_{ij}))\}$
or $\{\text{not}(\eta(s, \alpha_i)(B_n, P_{ij})) \text{ and } \text{not}(\eta(s', \alpha_i)(C_h, P_{ij}))\}$] ■

尚, 一般には, 初期状態の組から $\langle s, s' \rangle$ なる状態の組に到達する時には取り得ないレジスタ値の組が存在する. (2) はそのようなレジスタ値の組に対しても出

力が等しいことを要求しており, その意味で (1), (2) はマシンが等価であるための十分条件に相当する.

以下では, まず M_1, M_2 に相当する2つのグラフから同一の入力系列を与えたときに到達する状態の組を表す一つのグラフを構成する. 次に, このグラフ上で M_1, M_2 及び P 文 Ψ が (1), (2) を満たすための一つの十分条件を与える. さらにその十分条件が成り立つか否かの判定問題が決定可能であることを示す.

最初に, 準備として図 1.1 のようなグラフから次のようなグラフを作成する.

[死状態の導入と遷移の完全指定]

図 1.1 では, 全ての状態, レジスタ値, 入力組に対して遷移が指定されているわけではない. そこで, 特別の状態 (以下, 「死状態」と呼び, "se" で表す) を追加し, 図 1.1 のグラフから図 1.2 のようなグラフ (以下, 拡張グラフと呼ぶ) を作成する. 図 1.2 では, 図 1.1 の各状態 s と入力記号 α_i の組に対して, 定義域を指定する関数 $\eta(s, \alpha_i)(X_n, P_{ij})$ が恒真(true)でない限り $s \rightarrow se$ なる有向辺を加え, 4 字組

$\langle \alpha_i(P_{ij}), \text{not}(\eta(s, \alpha_i)(X_n, P_{ij})), -, - \rangle$

をその辺のラベルとする. 但し, 第 3, 4 引数の "-" は値を指定しないことを表す. 尚, 1.2.2 の定義より, $\eta(s, \alpha_i)(X_n, P_{ij})$ は P 文ゆえ恒真性は判定可能であり, 任意のグラフから拡張グラフを構成できる. ■

[例 2 プロトコルマシン M_2]

図 1.3 は, M_1 と等価と思われるマシン M_2 の拡張グラフである. M_2 の状態は, s_5, s_6, s_7 の3個で, それぞれ, 送信可能状態, 受信待状態, 最終状態を表し, 死状態を se で表している. レジスタは R_3, R_4, R_5 の3個で, R_3 の値が1の時は通常の送信のみを実行し, 0の時

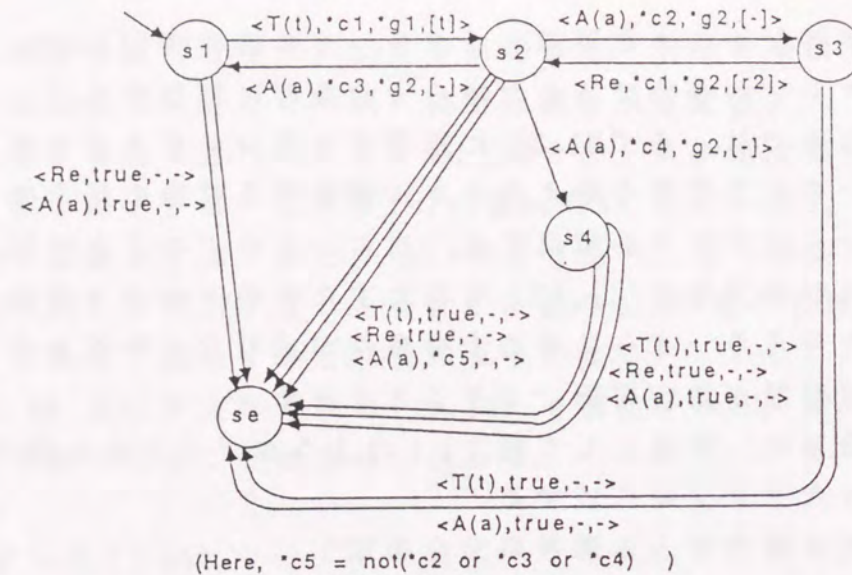


図 1.2 プロトコルマシン M1 の拡張グラフ

は再送のみを実行する。R4 は、送信回数を記憶するレジスタであり、初期値が10で、1回送信するごとに1ずつその値を減らす。R5 は、M1のR2同様、再送のためのデータを保持するために用いられる。■

次に、図 1.2, 1.3 のような2つのマシン M1, M2 の拡張グラフの組に同じ入力系列を与えた時に到達する状態の組を表すグラフ（「遷移グラフ」と呼ぶ）を作成する。

〔遷移グラフ〕

まず、M1, M2 の拡張グラフから、次のようなグラフ $G(V, E)$ を作成する。頂点集合 V は M1, M2 の状態の2字組 $\langle s, s' \rangle$ の集合とする。又、M1, M2 に各々 $s \rightarrow t$, $s' \rightarrow t'$ なる辺が存在し、且つその辺で同じ入力記号に対して、次の状態、レジスタ値、出力が指定されている（2つの辺のラベルの第1引数が同じである）時、且つその

$$M2 = \langle \{s5, s6, s7\}, \langle R3, R4, R5 \rangle, \langle s5, 1, 10, 0 \rangle, \{T, Re, A\}, \sigma, \delta', \rho' \rangle$$

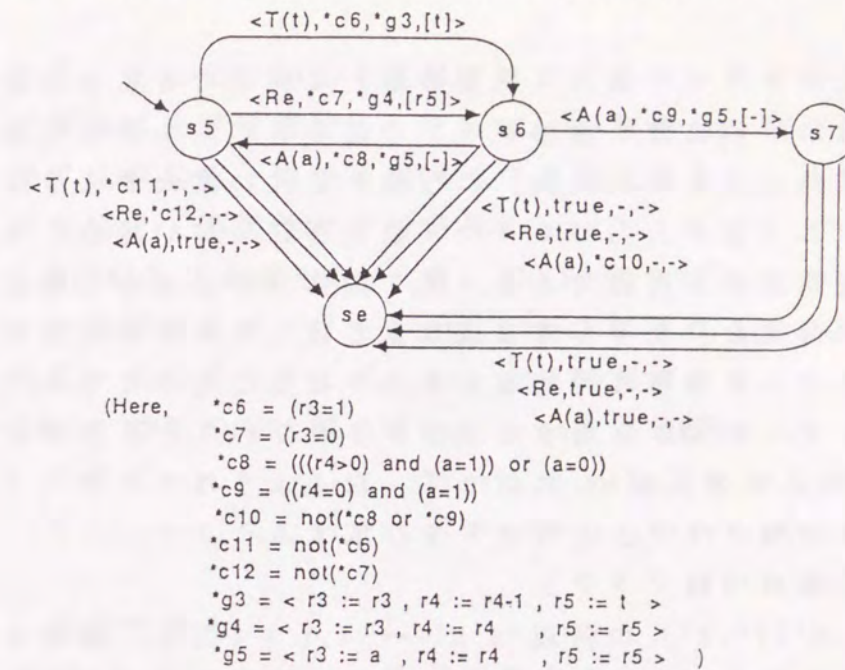


図 1.3 プロトコルマシン M2 の拡張グラフ

時のみ、辺 $\langle s, s' \rangle \rightarrow \langle t, t' \rangle$ を E の要素とする。又、辺 $s \rightarrow t$ 及び $s' \rightarrow t'$ のラベルが、各々 $\langle a_1, a_2, a_3, a_4 \rangle$, $\langle a_1, b_2, b_3, b_4 \rangle$ である時、 $\langle a_1, \{a_2, b_2\}, \{a_3, b_3\}, \{a_4, b_4\} \rangle$ を辺 $\langle s, s' \rangle \rightarrow \langle t, t' \rangle$ のラベルとする。さらに、上述のような方法で作成したグラフ $G(V, E)$ において、初期状態の2字組 $\langle s_1, s'_1 \rangle$ に相当する頂点からの有向道(path)が存在しない頂点や辺を取り除いたグラフ $G'(V', E')$ を作成し、そのグラフを M1, M2 に対する「遷移グラフ」と呼ぶ。例えば、図 1.4 は、図 1.2, 1.3 で定義したマシン M1, M2 に対する遷移グラフである。簡単のため、図 1.4 では各辺のラベルの第3, 4引数は省略している。■

遷移グラフの各頂点は、初期状態から到達する可能性のある状態の組を表している。尚、一般にはグラフ

上のすべての頂点（状態の組）に到達できるとは限らない。例えば、遷移グラフの辺のラベルの第2引数の条件式（上の $\{a_2, b_2\}$ に相当）を共に満たすようなレジスタ値や入力パラメータ値が存在しない場合、その辺は遷移不可能である。又、動作途中において検証者の与えたP文 Ψ が常に成り立てば、その条件を利用してさらに遷移不可能な辺を見つけることができる。そこで、M1, M2 に対する遷移グラフ $G'(V', E')$ 及びP文 $\Psi(u, v, B_n, C_h)$ に対して、次のようなグラフ（「遷移可能グラフ」と呼ぶ）を作成する。

[遷移可能グラフ]

$G'(V', E')$ の各辺 $\langle s, s' \rangle \rightarrow \langle t, t' \rangle$ （但し、辺のラベルを $\langle \alpha_1(P_{ij}), \{A2(B_n, P_{ij}), B2(C_h, P_{ij})\}, \{a_3, b_3\}, \{a_4, b_4\} \rangle$ とする）に対して、

$\forall B_n, C_h, P_{ij}$

$[\Psi(s, s', B_n, C_h)$

$\supset \text{not}\{A2(B_n, P_{ij}) \text{ and } B2(C_h, P_{ij})\}]$

--(*1)

が真（状態が s, s' の時、その時点のレジスタ値が Ψ を満足すれば、入力記号が α_1 であるようななどのような入力パラメータ値に対しても $\langle s, s' \rangle$ から $\langle t, t' \rangle$ に遷移しない）ならば、その辺を遷移グラフから取り除く。このような辺すべてを E' から取り除いた後、初期状態の2字組 $\langle s_1, s'_1 \rangle$ に相当する頂点からの有向道(path)が存在しない頂点や辺を取り除いたグラフ $G''(V'', E'')$ を作成し、このグラフをM1, M2, Ψ に対する「遷移可能グラフ」と呼ぶ。尚、1.2.2 の定義より状態 s, s' を整数に対応付けているので、 $\Psi(s, s', B_n, C_h)$ は B_n, C_h を変数とするP文となる。また、1.2.2 で述べた制約(2)より、 $A2(B_n, P_{ij}), B2(C_h, P_{ij})$ もそれぞれ

P文。よって、(*1)全体はプレスブルガー文となりその真偽は決定可能⁽⁵⁾。よって、任意のM1, M2及びP文 Ψ に対して、遷移可能グラフを作成できる。 ■

[命題1（等価性の十分条件）]

M1, M2及びP文 $\Psi(u, v, B_n, C_h)$ に対して、上で定義した遷移可能グラフを $G''(V'', E'')$ とする。この時、次の①～④が成り立てば、M1とM2は等価である。

① M1とM2の初期状態の組 $\langle s_1, s'_1 \rangle$ 及び初期レジスタ値の組 $\langle r_{11}, \dots, r_{1n} \rangle, \langle r'_{11}, \dots, r'_{1n} \rangle$ に対して、
 $\Psi(s_1, s'_1, \langle r_{11}, \dots, r_{1n} \rangle, \langle r'_{11}, \dots, r'_{1n} \rangle)$

--(*2)

が真である。

② $G''(V'', E'')$ には、一方が死状態でなく、他方が死状態であるような状態の組（頂点）は存在しない。

③ $G''(V'', E'')$ において、 E'' に含まれる任意の辺 $\langle s, s' \rangle \rightarrow \langle t, t' \rangle$ （但し、 t, t' は死状態でない。また、辺のラベルを $\langle \alpha_1(P_{ij}), \{a_2, b_2\}, \{A3(B_n, P_{ij}), B3(C_h, P_{ij})\}, \{a_4, b_4\} \rangle$ とする）に対して、

$\forall B_n, C_h, P_{ij}$

$[\Psi(s, s', B_n, C_h) \supset$

$\Psi(t, t', A3(B_n, P_{ij}), B3(C_h, P_{ij}))]$ --(*3)

が成り立つ。

④ $G''(V'', E'')$ において、 E'' に含まれる任意の辺 $\langle s, s' \rangle \rightarrow \langle t, t' \rangle$ （但し、 t, t' は死状態でない。また、辺のラベルを $\langle \alpha_1(P_{ij}), \{a_2, b_2\}, \{a_3, a_4\}, \{A4(B_n, P_{ij}), B4(C_h, P_{ij})\} \rangle$ とする）に対して、

$\forall B_n, C_h, P_{ij}$

$[\Psi(s, s', B_n, C_h) \supset$

$A4(B_n, P_{ij}) = B4(C_h, P_{ij})]$ --(*4)

が成り立つ。 ■

(1)の場合: $\hat{\eta}(w'), \hat{\eta}'(w')$ の値が共に偽となり,
(1), (3) の定義より長さが $k+1$ の w' に対する (1), (3)
が成り立つ.

(II)の場合: $\hat{\eta}(w'), \hat{\eta}'(w')$ の値が共に真となる.
状態の組 $\langle t, t' \rangle$ は上述のように $V'' - \{ \langle se, se \rangle \}$ の要素で
あり, 長さが $k+1$ の w' に対する (3) が成り立つ. 帰納
法の仮定 (1) より $\Psi(s, s', \delta(w), \delta'(w))$ の値が真で
あること, 及び ③ が成り立つことより状態の組 $\langle t, t' \rangle$
においても $\Psi(t, t', \delta(w'), \delta'(w'))$ の値が真であ
る. よって, 長さが $k+1$ の w' に対する (1) が成り立つ.

次に (1), (3) 及び ②, ④ から (2) が成り立つことを示
す. 状態の組 $\langle s, s' \rangle$ に対して $\hat{\sigma}(w) = s, \hat{\sigma}'(w) = s'$ な
る入力系列 w が存在しない場合は考える必要がない.
そのような入力系列 w が存在する場合, (3) より $\langle s, s' \rangle$
は $V'' - \{ \langle se, se \rangle \}$ の要素. 上述の議論と同じように (1)
及び ② を用いて, 任意のレジスタ値と入力 $\alpha_i (Q_{ij})$
の組に対して, 次の状態は共に死状態か共に死状態で
ないかのどちらかである. 共に死状態の場合, 上述の
(1) の場合同様 (2) の $\eta \langle s, \alpha_i \rangle (\dots), \eta' \langle s', \alpha_i \rangle (\dots)$
の値は共に偽となり (2) が成り立つ. 共に死状態でな
い場合, 上述の (II) の場合同様 (2) の $\eta \langle s, \alpha_i \rangle (\dots),$
 $\eta' \langle s', \alpha_i \rangle (\dots)$ の値は共に真となる. ④ が成り立つ
ので, そのようなレジスタ値と入力に対して出力は等
しく, 従って (2) が成り立つ. (証明終)

[命題 2]

$M1, M2$ 及び P 文 Ψ が与えられた時, 命題 1 の ① ~ ④
が成り立つかどうかは決定可能である. ■

(証明) 前述のように, 任意のマシンの組 $M1, M2$ 及び
 P 文 Ψ に対して, 遷移可能グラフが構成できる. 又,
 $\Psi(u, v, B_n, Ch)$ は P 文ゆえ (*2) は変数を含まない P
文. よって ① の真偽は決定可能. ② は遷移可能グラフ
を構成すれば判定可能. ③ については 1.2.2 の定義より
 s, s' は整数であり, (*3) の $\Psi(s, s', B_n, Ch)$ は, $B_n,$
 Ch を変数とする P 文. 又, 次のレジスタ値を指定す
る関数 $A3(B_n, P_{ij}), B3(Ch, P_{ij})$ も 1.2.2 で述べた制
約 (3) より, 各々 B_n と P_{ij} (又は Ch と P_{ij}) の 1 次
結合で表される. よって, $\Psi(t, t', A3(B_n, P_{ij}), B3$
 $(Ch, P_{ij}))$ も B_n, Ch, P_{ij} を変数とする P 文. ゆえ
に (*3) もプレスブルガー文となり, その真偽は決定可
能⁽⁵⁾. ④ についても ③ 同様出力を指定する関数 $A4(B$
 $n, P_{ij}), B4(Ch, P_{ij})$ が B_n と P_{ij} (または Ch と P_{i
 j) の 1 次結合で表される. よって (*4) もプレスブルガ
ー文となり, その真偽は決定可能⁽⁵⁾. (証明終)

第 1 章では, $M1, M2$ 及び P 文 $\Psi(u, v, B_n, Ch)$ に対
して命題 1 の ① ~ ④ が共に成り立つ時, 且つその時のみ
 $M1$ と $M2$ は Ψ 等価であると定義する. 尚, $\forall x_1, \dots, x_n (A)$
の形のプレスブルガー文の真偽は P 文 A の否定が整数
解を持たないかどうかを判定すればよく, 整数線形計
画問題の解の非存在性の判定手続きを用いて判定でき
る⁽⁶⁾.

次に, 図 1.2, 1.3 で与えた 2 つのマシン $M1, M2$
に対して, Ψ 等価となるような P 文 Ψ を与える.

$$\begin{aligned} \Psi(u, v, \langle r_1, r_2 \rangle, \langle r_3, r_4, r_5 \rangle) \\ = \{ (r_1 + r_4 = 10) \text{ and } (r_2 = r_5) \text{ and } (u = s_1 \supset r_3 = 1) \\ \text{and } (u = s_3 \supset r_3 = 0) \} \quad \text{--} (*5) \end{aligned}$$

(但し, u, v はそれぞれ $M1, M2$ の状態を表す変数, $r_1,$
 r_2 は $M1$ のレジスタ $R1, R2$ の値を表す変数, $r_3, r_4,$

r_5 は M2 のレジスタ R3, R4, R5 の値を表す変数)。

図 1.4 において、斜線で塗りつぶされた頂点は M1, M2 の遷移グラフから M1, M2, Ψ に対する遷移可能グラフを作成する際に取り除かれる (遷移不可能な) 頂点であり、斜線で塗りつぶされていない頂点とその間を結ぶ辺が、M1, M2, Ψ に対する遷移可能グラフを表している。さらに、図 1.2, 1.3 で与えた M1, M2 及び (*5) で指定した P 文 Ψ に対して、命題 1 の①～④が成り立つ。よって、M1 と M2 は Ψ 等価となり、従って M1 と M2 は等価である。

マシン M1, M2 及び P 文 Ψ を与えて、M1, M2 が Ψ 等価かどうかの判定は計算機を用いて機械的に行えるので、上の方法は、等価性を証明する際の有効な方法に成り得ると考えられる。

1.5 検証例

筆者らは、マシン M1, M2 及び P 文 Ψ を入力として、マシン M1, M2 の Ψ 等価性を判定するプログラムを作成し⁽³⁾⁽⁴⁾、データリンクレベルにおける SDLIC 手順の 2 次局⁽⁸⁾に相当する 2 つのマシンについて等価性の証明を行った。証明に使用したマシンの 1 つは、あるメーカーの作成した装置⁽⁹⁾をそのまま今のモデルで記述したもの (マシン A と呼ぶ) であり、もう 1 つのマシンは、マシン A と全く独立に筆者らが文献 (8) の仕様書を基に作成したもの (マシン B と呼ぶ) である。

入力記号は 12 個で、各入力記号は 0 ～ 4 個のパラメータを持つ。出力は整数の 8 字組である。2 つのマシン A, B の状態数は各々 11 個と 7 個、レジスタの数は各々 4 個と 6 個である。又、各マシンの仕様を図 1.

1 のようなグラフで表現した場合、それぞれの状態から最大 5 個 (平均 2 個) 程度の有向辺が出ている。

このような 2 つのマシン A, B に対して、マシン A, B の状態とレジスタ間で常に成り立つと思われる関係を 12 個抽出し、それら 12 個の関係が共に成り立つとして Ψ を指定した。これらの A, B, Ψ に対して、約 2000 個のプレスブルガー文の真偽を判定し、およそ 30 分の cpu 時間 (SUN3/60) で A, B が Ψ 等価であるという結果を得た。

尚、作成したプログラムでは判定時間を短縮するため、整数解の非存在性を調べる代わりに、実数解の非存在性を調べている。この方法では、整数解は存在しないけれども実数解が存在する場合は証明に失敗するが、 $x < y$ を $x \leq y - 1$ に変形する等の処置を講じて、できるだけ作成したプログラムでも証明が失敗しないよう配慮した。このため、筆者らが行った幾つかの検証例では、すべて上述の方法で証明が成功した。また、作成したプログラムでは、検証者が与えた Ψ に対して Ψ 等価であると判定出来なかった場合でも、遷移可能グラフを出力し、どの頂点 (状態の 2 字組) と入力記号の組が命題 1 の①～④を満足しなかったかを表示する。これらの表示からどのように Ψ を変更すれば、あるいは、どのように M1, M2 の仕様を変更すれば Ψ 等価となるか、についての有益な情報が得られる。

1.6 結言

第1章では、検証者が与えたP文 Ψ に対して、2つのプロトコルマシン間の Ψ 等価性を定義し、それらが Ψ 等価であるかどうかを判定する手続きを与えた。またその手続きを実際の装置例（SDLC手順の2次局）に適用し、「等価性」（従って「互換性」）の証明において、第1章の方法が有効であることを確かめた。尚、現在第1章で述べたモデルや検証法が等価性以外の検証の問題に適用出来るかどうかを検討している。

第2章

通信プロトコルにおける エラーリカバリ性の検証 の 一 方 法

2.1 序言

近年のデータ通信の発展に伴い、通信の手順を規定する通信プロトコル（以下、単にプロトコルと呼ぶ）の厳密かつ形式的な記述やその正しさの検証が重要な課題になっており、様々な研究が行われている⁽¹⁰⁾。

通常、プロトコルは、正常な状況（正常に送受信を行っているような状況）における各プロトコルマシン（通信制御装置）の動作を記述した部分と、異常な状況（回線に障害等が発生したような状況）からどのように正常な状況に復帰するかを記述した部分、に分類されることが多い。この際、「異常な状況からいつかは正常な状況に復帰すること」（エラーリカバリ性を持つという）を機械的に証明できれば望ましい。しかし、従来、エラーリカバリ性を証明する有効な方法がなく、テスト手法による検査が行われてきたが、このようなテスト手法では必ずしもエラーリカバリ性を論理的に保証できない。本論文では、プロトコルマシンの動作を記述するためのモデルを一つ定め、そのモデルで記述されたプロトコルマシン対がエラーリカバリ性を持つことを機械的に証明するための方法を提案する⁽¹²⁾。

第1章と同様に、各プロトコルマシンは、「有限制御部」と任意の整数値を保持する有限個の「レジスタ」を持つオートマトンとして記述し、このオートマトン上での演算を $+$, $-$, $=$, $>$, and , or , not に限定する。このモデルで記述した2つのプロトコルマシンM1, M2の状態とレジスタ値の組（以下、M1, M2対の全状態、または単に全状態と呼ぶ）を引数とする述語 Φ を線形不等式の論理結合の形で指定する。述語 Φ の値は、その値が真の時は「正常な状況」を表すように指定し、「M1,

M2の初期状態と初期レジスタ値の組から、M1とM2が（通信回線の誤り等も考慮して）送受信等の動作を行うことによって遷移する任意の全状態に対して、たとえその全状態が Φ を満足しない全状態であっても、それ以降可能などのような動作を続けてもいつかは再び Φ を満足する全状態に遷移する」時、M1, M2は Φ に対するエラーリカバリ性を持つと定義する。

提案するモデルのマシン及び Φ のクラスにおいては、M1, M2が Φ に対するエラーリカバリ性を持つ場合でも、そのことを証明するアルゴリズムが存在しない。そこで、必ずしも証明に成功する保証はないが、「 Φ を満足する全状態からどのような動作を行っても再び Φ を満足する全状態へ遷移すること」を保証するための一つの手続きを与え、その手続きを再帰的に用いることによって、M1, M2が Φ に対するエラーリカバリ性を持つことを証明する。本章では、第1章と同様に、M1, M2が Φ に対するエラーリカバリ性を持つことを整数線形計画問題の解の非存在性の証明に帰着して機械的に証明している。またその証明手続きを用いて、HDLC手順⁽¹⁴⁾に従うあるプロトコルマシン対のエラーリカバリ性を検証することにより、提案する方法の有効性を示した。

2.2 プロトコルのモデル化

本章では、第1章と同様に、プロトコルマシンを「有限制御部」及び任意の整数値を保持する有限個の「レジスタ」を持つオートマトンとして記述する。また、「相手局からのフレームの受信」や「タイムアウトの発生」、「相手局へのフレームの送信」等を各々オー

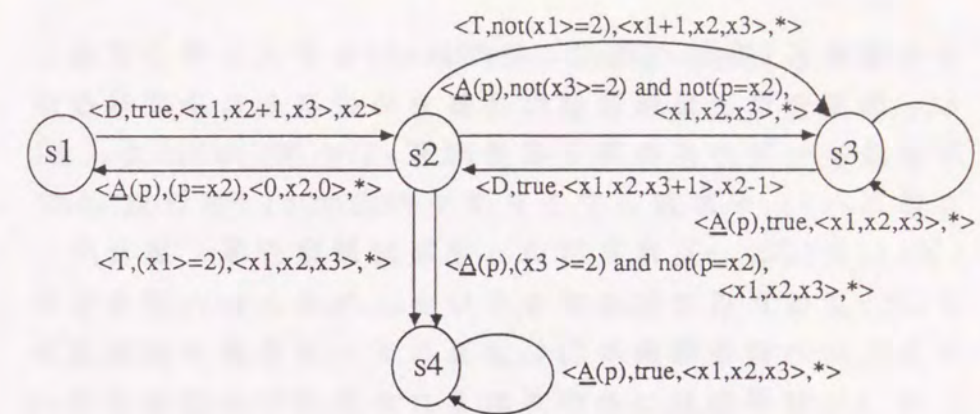
トマツンの「入力記号」とみなす。入力記号はその動作の種類により、

- (1) 相手局へのフレームの送信動作に相当する入力記号の有限集合(送信動作集合) I_s 。
- (2) 相手局からのフレームの受信動作に相当する入力記号の有限集合(受信動作集合) I_r 。
- (3) その局内で単独に行われる動作に相当する入力記号の有限集合(単独動作集合) I_o 。

に分類する。例えば、フレームの送信や再送は I_s に、受信は I_r に、タイムアウトの発生は I_o に属する。送信動作集合 I_s に属する入力記号に対してのみ「出力」が定義される。出力は整数とし、相手局へ送信されるフレームに相当する。また、受信動作集合 I_r に属する入力記号のみ「パラメータ」を持ってもよい。パラメータ値は整数とし、通信相手局から受信したフレームに相当する。

プロトコルマシンは、現在の状態(以下、状態は有限制御部の状態を表す)、レジスタ値及び入力(入力記号とその入力記号のパラメータ値の組)の組に対して、次の状態、次のレジスタ値、出力が定義されるいわゆる Mealy 型の決定性オートマトンとしてモデル化する。モデル化したプロトコルマシンの仕様は、図 2.1, 2.2 (例 1, 2 参照) のような有向グラフで記述する。

図において、グラフの頂点はプロトコルマシンの状態を表し、有向辺及びそのラベルによって各入力に対する次の状態、次のレジスタ値、出力を指定する。グラフの各辺 $s \rightarrow s'$ には、4 字組のラベル $\langle$ 入力, 条件式, 次のレジスタ値, 出力 $\rangle$ が付加されており、その第 1 成分(入力)に書かれた入力記号 α が $I_s, I_r,$



States s1: Initial state (Transmission state)
s2: Ack-waiting state
s3: Retransmission state
s4: Operator-notification state

Registers x1: The number of Timeout (Initial value=0)
x2: The number of Transmission (Initial value=0)
x3: The number of Retransmission (Initial value=0)

Input symbols D: Transmission
A: Reception of acknowledgement (p: parameter of "A")
T: Timeout
 $I_s = \{D\}, I_r = \{A\}, I_o = \{T\}$

図 2.1 プロトコルマシン M1

I_o の何れに属するかに応じて各々①～③のようなラベルが与えられる。

- ① $\langle \alpha, c(X_n), \delta(X_n), \rho(X_n) \rangle \quad (\alpha \in I_s)$
- ② $\langle \alpha(p), c(X_n, p), \delta(X_n, p), * \rangle \quad (\alpha \in I_r)$
- ③ $\langle \alpha, c(X_n), \delta(X_n), * \rangle \quad (\alpha \in I_o)$

ここで、 n はレジスタの数、 X_n は現在のレジスタ値を表す変数 $x_1, x_2, \dots, x_n$ の組を表す。 p は入力記号 α のパラメータ値を表す変数である。" $2x+3y+10$ " のように整数と変数の線形結合を P 項と呼び、" $\text{not}(y=w+1)$ and $(x+y>z-1)$ " のように、 P 項, $=, >, \text{and}, \text{or}, \text{not}$ から

なる論理式(線形不等式の論理結合)をP文と呼ぶ(但し、以下では表現を簡単にするため便宜上 $\geq$ や $\sqsubset$ も使用する)。ラベルの第2成分 $c(X_n), c(X_n, p)$ は X_n (または X_n, p)を変数とするP文で指定する。第3成分 $\delta(X_n), \delta(X_n, p)$ はP項のn字組で指定する。また、 $\rho(X_n)$ はP項で指定する。いま、与えられたプロトコルマシンの仕様中に①のようなラベルを持つ有向辺 $s \rightarrow s'$ が存在し、そのプロトコルマシンの現在の状態がs、その時のレジスタ値の組が R_n 、入力 $\alpha (\in I_s)$ であるとする。この時、 $c(R_n)$ の値が真ならば、次の状態は s' 、次のレジスタ値の組は $\delta(R_n)$ となり、 $\rho(R_n)$ が相手局に出力される。また、②、③のようなラベルを持つ有向辺の場合、出力は指定されないで、ラベルの第4成分を*としている。尚、一つの頂点からラベルの第1成分の入力が同じであるような有向辺が2本以上出てもよいが、それらの辺のラベルの第2成分の条件式は、オートマトンが決定性に動作するように指定されなければならない。

[例1 プロトコルマシンM1]

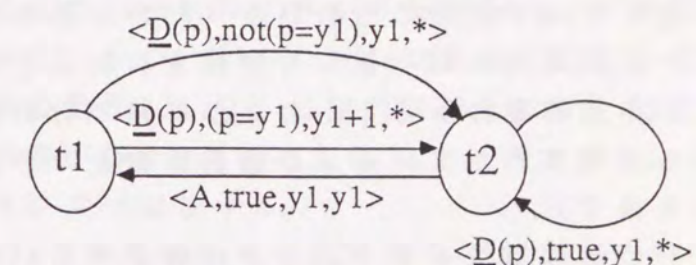
図2.1のプロトコルマシンM1はデータリンクレベルにおけるHDLC手順の一次局を一部変更し、かつ簡単化したものである。このマシンは、HDLC手順におけるウィンドウサイズを1に制限したモデルであり、フレームを1個送信(動作D)し、それに対する応答フレームを受信(動作A)するという動作を繰り返す。もし、フレームの送信後にタイムアウト(動作T)が発生した場合、フレームが正しく受信されなかったと判断し、そのフレームの再送(動作D)を行う。タイムアウトの発生回数(レジスタ X_1 の値)が2以上、またはフレームの再送回数(レジスタ X_3 の値)が2以上とな

った場合には、オペレータによる回復を必要とする特別の状態 s_4 (オペレータ通告状態と呼ぶ)へ遷移する。尚、レジスタ X_2 は、新規フレームの送信回数を保持するために用いられる。■

例えば、現在の状態が s_2 、レジスタ X_1, X_2, X_3 の値が各々 r_1, r_2, r_3 、入力 T の場合、 r_1 が2未満ならば状態 s_3 に遷移し、 X_1, X_2, X_3 の値は各々 r_1+1, r_2, r_3 となる。 r_1 が2以上ならば状態 s_4 に遷移するが、レジスタ X_1, X_2, X_3 の値は変化しない。尚、入力記号 T は1。に属するので出力は指定されない。

[例2 プロトコルマシンM2]

図2.2のプロトコルマシンM2はデータリンクレベルにおけるHDLC手順の二次局を簡単化したものであり、図1のM1と送受信を行う。M2には相手局への応答フレームの送信(動作A)と相手局からのフレームの受信(動作D)という2つの動作があり、相手局から受信したフレームの個数を保持するレジスタ Y_1 がある。■



States t1: Initial state (Reception state)
t2: Ack-transmission state

Register y1: The number of Reception (Initial value=0)

Input symbols D: Reception (p: parameter of "D")
A: Transmission of acknowledgement
 $I_s = \{A\}, I_r = \{D\}, I_o = \{\}$

図2.2 プロトコルマシンM2

2.3 通信系の動作

本論文では互いに通信を行う2つのプロトコルマシン対から成る通信系を考え、次のような仮定の下でエラーリカバリ性を議論する。

- (A) フレームを送信する場合、そのフレームが送信と同時に相手局に正しく受信されるか、あるいは、送信と同時に通信回線上で消失するかの何れかである。
- (B) 受信が正しく行われた場合、送信局の送信動作によって出力される整数値が受信局の受信動作に相当する入力記号のパラメータ値として受け渡される。

以上の仮定より、プロトコルマシン対の動作として次の6つの動作を考え、これらを通信系の動作と呼ぶ。尚、M1, M2の送信動作集合を I_{s1}, I_{s2} 、受信動作集合を I_{r1}, I_{r2} 、単独動作集合を I_{o1}, I_{o2} 、で各々表す。[通信系の動作]

- (1) M1の送信動作 $\alpha \in I_{s1}$ と α に対応するM2の受信動作 $\underline{\alpha} \in I_{r2}$ が同時に行われる(M1から送信されたフレームが同時にM2へ正しく受信されたことを表す)。
- (2) M1の送信動作 $\alpha \in I_{s1}$ のみが単独で行われる(M1から送信されたフレームが通信回線上で消失したことを表す)。
- (3) M2の送信動作 $\beta \in I_{s2}$ と β に対応するM1の受信動作 $\underline{\beta} \in I_{r1}$ が同時に行われる。
- (4) M2の送信動作 $\beta \in I_{s2}$ のみが単独で行われる。
- (5) M1の単独動作集合 I_{o1} に属する動作 γ が単独で行われる。
- (6) M2の単独動作集合 I_{o2} に属する動作 γ が単独で行われる。 ■

本論文では(1)~(6)の通信系の動作を各々 $[\alpha, \underline{\alpha}]$, $[\alpha, *]$, $[\underline{\beta}, \beta]$, $[*, \beta]$, $[\gamma, *]$, $[*, \gamma]$ で表す。又、 $[\alpha, \underline{\alpha}][\underline{\beta}, \beta]$ については、M1(M2)から $\alpha(\beta)$ によって出力される整数値が、M2(M1)の $\underline{\alpha}(\underline{\beta})$ の入力パラメータ値として割り当てられるとする。

例えば、図2.1, 2.2のプロトコルマシンM1, M2において、M1, M2の状態が各々 s_1, t_1 で、レジスタ X_1, X_2, X_3 の値が各々0, 3, 0で、レジスタ Y_1 の値が3であったとする。M1で $\langle D, \text{true}, \langle x_1, x_2+1, x_3 \rangle, x_2 \rangle$ なるラベルを持つ有向辺 $s_1 \rightarrow s_2$ が存在するので、通信系の動作 $[D, *]$ (M1がフレームの送信を行ったがそのフレームが回線上で消失したことを表す)が実行可能となり、 $[D, *]$ の実行後、M1の状態のみが s_1 から s_2 に変化し、レジスタ X_2 の値が1増加する。

さらに、M2では $\langle \underline{D}(p), (p=y_1), y_1+1, * \rangle$ なるラベルの有向辺 $t_1 \rightarrow t_2$ が存在する。M1の出力(レジスタ X_2 の値)3をM2の入力記号 $\underline{D}$ のパラメータ(変数 p)に割り当ててM2の $t_1 \rightarrow t_2$ のラベルの条件($p=y_1$)を評価すると真となり、かつ、M1の $s_1 \rightarrow s_2$ の条件も真であるので、通信系の動作 $[D, \underline{D}]$ は実行可能となり、実行後、M1, M2の状態が各々 s_2, t_2 に変化し、レジスタ X_2, Y_1 の値が各々1ずつ増加する。

尚、M1, M2の状態対を一つの状態とみなし、(1)~(6)のような通信系の動作を入力記号とみなすと、プロトコルマシン対全体を一つのオートマトン(有限制御部と有限個のレジスタから成るMealy型のオートマトン)とみなすことができる。以下では、このオートマトンをM1, M2の対オートマトンと呼ぶ。また、 $\langle s_1, t_1, \langle 0, 3, 0 \rangle, \langle 0 \rangle \rangle$ のように、M1, M2の状態とレジスタ値の組をM1, M2対の全状態(あるいは単に全状態)と呼ぶ。

2.4 エラーリカバリ性の定義

まず、2つのプロトコルマシンM1, M2対の全状態を引数とする述語 Φ をP文で指定する。述語 Φ の値は、その値が真の時は「正常な状況」を表すように検証者が指定する。例えば、図2.1, 2.2のプロトコルマシンM1, M2に対して、M1, M2対が次の(1)または(2)を満足する時、正常な状況であるとみなすことにする。

(1) M1から送信された新規フレームの個数(レジスタ X_2 の値)とM2が正しく受信したフレームの個数(レジスタ Y_1 の値)が等しく、且つ最後に送信したフレームに対するタイムアウトの発生回数(レジスタ X_1 の値)や再送回数(レジスタ X_3 の値)が共に0である。

(2) M1がオペレータによるリカバリを必要とする特別な状態(オペレータ通告状態 s_4)にいる*。

この場合、次のような述語 Φ を指定すればよい**。

脚注+: 通常、状態 s_4 に遷移した場合、オペレータが相手局のオペレータと協調してM1, M2の状態やレジスタ値を初期化するという回復動作が行われ、それにより、エラーから「リカバリ」したと考えるのが妥当である。しかし、本論文では、簡単のため、 s_4 に遷移することによって「リカバリ性」を持つというような表現を用いる。

脚注+: 本論文では、状態 $s_0, s_1, s_2, \dots$ を各々整数(例えば0, 1, 2, ...)とみなすことにする。その場合、 Φ 中の $(u_1 = s_4)$ はP文となり、 Φ もP文となる。

$$\begin{aligned} \Phi(u_1, u_2, x_1, x_2, x_3, y_1) = & \\ \{ (x_2 = y_1) \text{ and } (x_1 = 0) \text{ and } (x_3 = 0) \} & \\ \text{or } (u_1 = s_4) & \dots (i) \\ (u_1, u_2 \text{ は各々 M1, M2 の状態を表す変数. } x_1, x_2, x_3 \text{ は} & \\ \text{各々 M1 のレジスタ } X_1, X_2, X_3 \text{ の値を表す変数.} & \\ y_1 \text{ は M2 のレジスタ } Y_1 \text{ の値を表す変数.}) & \end{aligned}$$

[エラーリカバリ性の定義]: M1, M2の初期状態と初期レジスタ値の組から2.3で述べた通信系の動作を行うことによって遷移する任意の全状態に対して、たとえその全状態が Φ を満足しない全状態であっても、それ以降可能などのような通信系の動作を続けても、いつかは再び Φ を満足する全状態に遷移する時、M1, M2は Φ に対するエラーリカバリ性を持つという。■

一般に、異なる Φ を指定することによって、異なるエラーリカバリ性を議論できる。

エラーリカバリ性の定義より、M1, M2が Φ に対するエラーリカバリ性を持てば、M1, M2は時々 Φ を満足しながら無限に遷移し続けるか、あるいは、遷移を終了する時は必ず Φ を満足する全状態で終了する。

[命題1]⁽¹⁵⁾

2.2で述べたプロトコルマシン及び Φ のクラスに対しては、M1, M2が Φ に対するエラーリカバリ性を持つ場合でも、そのことを証明するアルゴリズムは存在しない。■

2.5 エラーリカバリ性の検証法

命題1が成り立つので、以下では、必ずしも証明に成功するとは限らないが、もし、成功すればエラーリカバリ性を保証する一つの証明手続きを与える。

まず幾つかの定義を行う。図2.1, 2.2の $\langle x_1, x_2 + 1, x_3 \rangle$ や $\langle y_1 + 1 \rangle$ のようにM1, M2のレジスタ値を計算するためのP項の組を各々M1, M2のレジスタ計算式と呼び、

$$\langle s_2, t_2, \langle x_1, x_2 + 1, x_3 \rangle, \langle y_1 + 1 \rangle \rangle$$

のように、M1, M2の状態対とM1, M2のレジスタ計算式の4字組を Σ 式と呼ぶ。また、特別な場合として、

$$\langle s_1, t_1, \langle x_1, x_2, x_3 \rangle, \langle y_1 \rangle \rangle$$

のように、M1, M2のレジスタ計算式が変数のみからなる Σ 式を特に Σ 項と呼ぶ。

図2.1, 2.2のような2つのプロトコルマシンM1, M2及び $\langle s_1, t_1, \langle x_1, x_2, x_3 \rangle, \langle y_1 \rangle \rangle$ のようなM1, M2の一つの Σ 項が与えられると、図2.3のように、与えられた Σ 項からM1, M2の状態やレジスタ値がどのように変化していくかを表す(無限の大きさの)グラフ(木)を考えることが出来る。すなわち、図2.3の根ノードにはM1, M2の一つの Σ 項 $\langle s_1, t_1, \langle x_1, x_2, x_3 \rangle, \langle y_1 \rangle \rangle$ がラベルとして付加されており、各枝には通信系の動作と条件式の対がラベルとして付加されている。各ノードには Σ 式がラベルとして付加されており、根ノードからそのノードに至るまでの枝に付加された通信系の動作を順次実行した時に到達する状態対とその時点のレジスタ値が根ノードのレジスタ値のどのような関数で表されるかが記述されている。例えば、図2.3の

$$\langle s_1, t_1, \langle x_1, x_2, x_3 \rangle, \langle y_1 \rangle \rangle$$

$$\rightarrow \langle s_2, t_1, \langle x_1, x_2 + 1, x_3 \rangle, \langle y_1 \rangle \rangle$$

$$\rightarrow \langle s_3, t_1, \langle x_1 + 1, x_2 + 1, x_3 \rangle, \langle y_1 \rangle \rangle$$

という道を辿ることによって、 $[D, *]$, $[T, *]$ なる動作を実行した時に状態対とレジスタ値がどのように変化するかがわかる。

以下では、M1, M2及び一つの Σ 項 $\langle s_a, t_a, X_n, Y_m \rangle$ (但し、 X_n, Y_m は各々M1, M2のレジスタ値を表す変数の組)が与えられた時、 $\langle s_a, t_a, X_n, Y_m \rangle$ を根ノードのラベルとする(一般に無限の大きさの)木が次のような条件を満足する時、その木を $\langle s_a, t_a, X_n, Y_m \rangle$ に対する遷移グラフと呼ぶ。

[遷移グラフの条件]

与えられた木のすべてのノード $\langle s, t, B_n, C_m \rangle$ (B_n, C_m は各々 X_n, Y_m を変数とするM1, M2のレジスタ計算式)に対して、次の①~④の条件が成り立つ。

① α がM1の送信動作又は単独動作でM1に $\langle \alpha, c_1(X_n), \delta_1(X_n), \rho_1(X_n) \rangle$ なるラベルの有向辺 $s \rightarrow s'$ が存在する時、且つその時のみ $\langle s', t, \delta_1(B_n), C_m \rangle$ なるラベルのノードが存在し、 $\langle [\alpha, *], c_1(B_n) \rangle$ なるラベルの枝

$$\langle s, t, B_n, C_m \rangle \rightarrow \langle s', t, \delta_1(B_n), C_m \rangle$$

が存在する。

② α がM1の送信動作で $\underline{\alpha}$ がM2の受信動作で、M1に $\langle \alpha, c_1(X_n), \delta_1(X_n), \rho_1(X_n) \rangle$ なるラベルの有向辺 $s \rightarrow s'$ が存在し、M2に $\langle \underline{\alpha}(p), c_2(Y_m, p), \delta_2(Y_m, p), * \rangle$ なるラベルの有向辺 $t \rightarrow t'$ が存在する時、且つその時のみ $\langle s', t', \delta_1(B_n), \delta_2(C_m, \rho_1(B_n)) \rangle$ なるラベルのノードが存在し、 $\langle [\alpha, \underline{\alpha}], \{c_1(B_n) \text{ and } c_2(C_m, \rho_1(B_n))\} \rangle$ なるラベルの枝

$$\langle s, t, B_n, C_m \rangle$$

$$\rightarrow \langle s', t', \delta_1(B_n), \delta_2(C_m, \rho_1(B_n)) \rangle$$

が存在する。

③ β がM2の送信動作又は単独動作でM2に $\langle \beta, c_2(Y_m), \delta_2(Y_m), \rho_2(Y_m) \rangle$ なるラベルの有向辺 $t \rightarrow t'$ が存在する時、且つその時のみ $\langle s, t', B_n, \delta_2(C_m) \rangle$ な

るラベルのノードが存在し、 $\langle [*, \beta], c_2(C_m) \rangle$ なるラベルの枝

$$\langle s, t, B_n, C_m \rangle \rightarrow \langle s, t', B_n, \delta_2(C_m) \rangle$$

が存在する。

- ④ β が M1 の受信動作で β が M2 の送信動作で、M1 に $\langle \beta(p), c_1(X_n, p), \delta_1(X_n, p), * \rangle$ なるラベルの有向辺 $s \rightarrow s'$ が存在し、M2 に $\langle \beta, c_2(Y_m), \delta_2(Y_m), \rho_2(Y_m) \rangle$ なるラベルの有向辺 $t \rightarrow t'$ が存在する時、且つその時のみ $\langle s', t', \delta_1(B_n, \rho_2(C_m)), \delta_2(C_m) \rangle$ なるラベルのノードが存在し、 $\langle [\beta, \beta], \{c_1(B_n, \rho_2(C_m)) \text{ and } c_2(C_m)\} \rangle$ なるラベルの枝 $\langle s, t, B_n, C_m \rangle \rightarrow \langle s', t', \delta_1(B_n, \rho_2(C_m)), \delta_2(C_m) \rangle$ が存在する。 ■

いま、 Σ 項 $\langle s_0, t_0, X_n, Y_m \rangle$ に対する遷移グラフの中に $\langle [a, b], c(B_n, C_m) \rangle$ なるラベルを持つ枝

$$\langle s, t, B_n, C_m \rangle \rightarrow \langle s', t', B'_n, C'_m \rangle$$

が存在したとする。この時、条件(a)

$$\forall X_n, Y_m$$

$[\Phi(s_0, t_0, X_n, Y_m) \supset \text{not}(c(B_n, C_m))] \dots (a)$ の値が真ならばこの枝を「遷移する可能性のない枝」と呼び、(a) の値が偽ならばこの枝を「遷移する可能性のある枝」と呼ぶ。

また、 $\langle s_0, t_0, X_n, Y_m \rangle$ に対する遷移グラフの各ノード $\langle s, t, B_n, C_m \rangle$ に対して、条件(b)

$$\forall X_n, Y_m$$

$[\Phi(s_0, t_0, X_n, Y_m) \supset \Phi(s, t, B_n, C_m)] \dots (b)$ の値が真ならば、ノード $\langle s, t, B_n, C_m \rangle$ を Φ 復帰ノードと呼ぶ。 $\langle s, t, B_n, C_m \rangle$ が Φ 復帰ノードならば、根

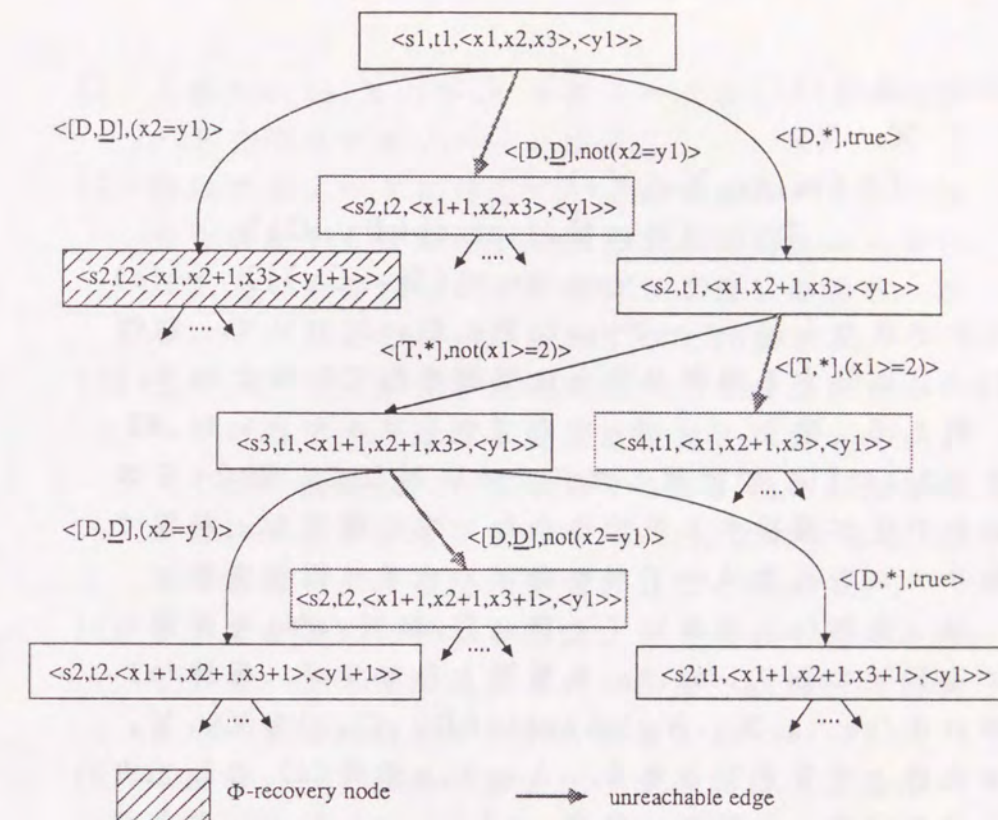


図 2.3 状態とレジスタ値の変化を表わすグラフ

ノードからこのノードに至る枝に付加された通信系の動作を順次実行することによって、全状態 $\langle s_0, t_0, R_n, V_m \rangle$ から全状態 $\langle s, t, R'_n, V'_m \rangle$ に遷移した時、もし前者の全状態が Φ を満足しているならば、後者の全状態もまた Φ を満足する。

さらに $\langle s_0, t_0, X_n, Y_m \rangle$ に対する遷移グラフの各ノード $\langle s, t, B_n, C_m \rangle$ から $\langle [a, b], c_1(B_n, C_m) \rangle, \langle [a, b], c_2(B_n, C_m) \rangle, \dots, \langle [a, b], c_p(B_n, C_m) \rangle$ のようにラベルの第 1 成分の動作が同じであるような枝（動作 $[a, b]$ に対する枝と呼ぶ）が p 本出ていたとする。こ

の時, 条件(c)

$\forall X_n, Y_n$

$[\Phi(s_0, t_0, X_n, Y_n)$

$\supset \{c_1(B_n, C_n) \text{ or } c_2(B_n, C_n)$

$\text{or } \dots \text{ or } c_p(B_n, C_n)\}] \dots (c)$

の値が真ならば, ノード $\langle s, t, B_n, C_n \rangle$ において, 動作 $[a, b]$ に対する遷移が完全に定義されているという。

例えば, 図2.1, 2.2のプロトコルマシンM1, M2及び2.4の(i)で定義したP文 Φ に対して, 図2.3の破線の枝が遷移する可能性のない枝に相当し, 斜線で塗りつぶされたノードが Φ 復帰ノードに相当する。

尚, 条件(a)において, B_n, C_n は X_n, Y_n を変数とするP項であり, s_0, t_0 を整数とみなせば, 条件(a)中の $\Phi(s_0, t_0, X_n, Y_n)$ も $\text{not}(c(B_n, C_n))$ も X_n, Y_n を変数とするP文となる。よって, 条件(a)の[]内全体もP文。一般に, P文, and, or, not, $\forall, \exists$ からなる論理式の内, その論理式に現れるすべての変数が $\forall, \exists$ で束縛されているような論理式をプレスブルガー文と呼び, その真偽は決定可能であることが知られている⁽¹¹⁾。上述の条件(a)全体は, $\exists$ を含まない

$\forall x_1, x_2, \dots, x_n [\Psi(x_1, x_2, \dots, x_n)]$

の形のプレスブルガー文である。その真偽判定については2.6参照。同様に条件(b), (c)の真偽も判定可能である。

[Φ -到達可能性グラフ]

P文 Φ , M1, M2及びM1, M2の状態対 $\langle s_0, t_0 \rangle$ が与えられて, 次のような手続きを実行し, (4)で手続きが終了した時, かつその時のみ, 手続き終了時に作成されたグラフ(木)を $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフと定義する。

- (1) Σ 項 $\langle s_0, t_0, X_n, Y_n \rangle$ を根ノードとし, まず根ノードを注目するノードとする。
- (2) 注目するノードに対して, 上述の遷移グラフの条件を満足する枝の中から遷移する可能性のある枝(上述の条件(a)の値が偽になる枝)を選び, 新しいノードとそれらの枝をすべて付加する。
- (3) 上の(2)で付加された枝を調べ, その注目するノードにおいて, 遷移が完全に定義されていない(上述の条件(c)の値が偽である)動作が存在する時は手続きを終了する($\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフの作成に失敗したと考える)。
- (4) すべての葉ノードが Φ 復帰ノードなら手続きを終了する。その時に生成された木を $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフと定義する。
- (5) Φ 復帰ノード以外の葉ノードがあれば, 任意の一つを注目するノードとし, (2)へ。 ■

尚, この手続きでは, 手続きが永久に終了しないこともあり, その場合も $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフの作成に失敗したと考える。定義より, 上述の手続きで Φ -到達可能性グラフの作成に成功した場合, (5)で選ぶ葉ノードの順に依存せず, 同じ到達可能性グラフが得られる。逆に, (3)で終了したとき, 葉ノードの選ぶ順にかかわらず, Φ -到達可能性グラフの作成は成功しない。

[補題1]

上の手続きでM1, M2の状態対 $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフが作成できれば, Φ を満足する任意の全状態 $\langle s_0, t_0, R_n, V_n \rangle$ からどのような通信系の動作を行っても, いつかは Φ を満足する全状態に遷移する。(証明) Φ -到達可能性グラフTの各ノードのレジス

タ計算式の変数 X_n, Y_m に R_n, V_m を代入した木を T' とする。 T では、各動作に対する遷移が完全に定義されているので、全状態 $\langle s_0, t_0, R_n, V_m \rangle$ から順次通信系の動作を実行した時に遷移する全状態の列は、必ず T' の根から一つの葉への道の上に書かれている。 T 中のすべての葉ノードが Φ 復帰ノードであるので、 $\langle s_0, t_0, R_n, V_m \rangle$ からどのような通信系の動作を行っても、いつかは Φ を満足する全状態に遷移する。 ■

一般に、 $M1, M2$ 及びその状態対 $\langle s_0, t_0 \rangle, P$ 文 Φ 、自然数 k が与えられた時、状態対 $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフでサイズが k 以下のものがあるか否かは決定可能である。以下、状態対 $\langle s_0, t_0 \rangle$ に対する Φ -到達可能性グラフの葉ノード (Φ 復帰ノード) $\langle s, t, R_n, V_m \rangle$ の状態対 $\langle s, t \rangle$ の集合を $F(\langle s_0, t_0 \rangle, \Phi)$ で表す。本論文では、次のようなエラーリカバリ性の検証手続きを提案する。

[命題2 (エラーリカバリ性の検証手続き)]

$M1, M2$ の初期状態と初期レジスタ値の組が Φ を満足する時、 $M1, M2, P$ 文 Φ 及び自然数 k を与えて、次の手続きを行い、(3.2) で手続きを終了すれば、 $M1, M2$ は Φ に対するエラーリカバリ性を持つ。

(1) $S \leftarrow \{ \langle s_1, t_1 \rangle \}$

(s_1, t_1 は $M1, M2$ の初期状態。 $\langle s_1, t_1 \rangle$ はマークされていない)

(2) S 中でマークされていない状態対 $\langle s, t \rangle$ を見つけ、上述の Φ -到達可能性グラフの作成手続きを用いて k 以下のサイズで $\langle s, t \rangle$ に対する Φ -到達可能性グラフが作成できるか否かを調べる。

(2.1) k 以下で作成できなければ終了 (証明に失敗)。

(2.2) k 以下で作成できれば、 $\langle s, t \rangle$ をマークし、

$S \leftarrow S \cup \{ \langle s', t' \rangle \mid \langle s', t' \rangle \in F(\langle s, t \rangle, \Phi),$
且つ $\langle s', t' \rangle \notin S \}$

(但し、 $\langle s', t' \rangle$ にはマークしない)

(3) S 中でマークされていない状態対が存在するか否かを調べる。

(3.1) 存在すれば、(2)へ。

(3.2) 存在しなければ終了 (証明に成功)。

(証明) (3.2) で手続きを終了した時の変数 S に初期状態の対が含まれること及び補題1よりあきらか。 ■

例えば、図2.1, 2.2のプロトコルマシン $M1, M2$ において、 Φ として2.4で述べた(i)の式を、 k の値として8以上の自然数を与えて命題2の手続きを実行すると、(3.2) で手続きが終了し、エラーリカバリ性の証明に成功する。また、4つの状態対 $\langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle, \langle s_4, t_1 \rangle, \langle s_4, t_2 \rangle$ が手続き終了時の変数 S に含まれる。尚、 Φ -到達可能性グラフを生成する場合、最悪深さ k の木が生成されるため、命題2のステップ(2)につき、一般には $O(C^k)$ 回 (k 以外は固定と考える。 C は定数) 条件(a)~(c)のようなプレスブルガー文の真偽判定を行わねばならない。

また、たとえ $M1, M2$ が Φ に対するエラーリカバリ性を持つ場合でも命題2の方法で証明に失敗することもある。しかし、このような場合でも、 $M1, M2$ の全状態を引数とするような適当な P 文 Q を見つけて、 $M1, M2$ が (Φ and Q) に対するエラーリカバリ性を持つことを証明することが出来る場合がある。その時は、もちろん、 $M1, M2$ が Φ に対するエラーリカバリ性を持つ。例えば、図2.1, 2.2のプロトコルマシン $M1, M2$ において、 Φ として、

$$\begin{aligned}\Phi(u_1, u_2, x_1, x_2, x_3, y_1) \\ = (x_2 = y_1) \text{ or } (u_1 = s_4) \quad \dots(ii)\end{aligned}$$

を指定し、 k の値を十分大きく取って命題 2 の手続きを実行しても、 k 以下のサイズで初期状態対 $\langle s_1, t_1 \rangle$ に対する Φ -到達可能性グラフを作成できないので、証明に失敗する。しかし、 Q として $M1$ のレジスタ x_1 と x_3 の値は 0 以上であるという性質、

$$\begin{aligned}Q(u_1, u_2, x_1, x_2, x_3, y_1) \\ = (x_1 \geq 0) \text{ and } (x_3 \geq 0) \quad \dots(iii)\end{aligned}$$

を指定し、命題 2 の手続きを用いて (Φ and Q) に対するエラーリカバリ性の証明を行うと証明に成功する。このように、 Q は Φ に対するエラーリカバリ性をいわゆる並行帰納法を用いて証明する時の補題に相当する。尚、初期状態と初期レジスタ値の組が Φ を満足しない場合でも、初期状態と初期レジスタ値の組から遷移可能な全状態を順次生成しながら初めて Φ を満足する全状態の集合を求め、それらの全状態に含まれる状態対の集合を命題 2 の (1) の変数 S の初期値として与えれば、同様の方法で扱える。

2.6 検証例

2.5 の命題 2 の手続きに相当するプログラムをワークステーション Sun-3/260 (メモリ 16MB) 上で言語 C を用いて作成した。作成したプログラムでは、各状態対に対する到達可能性グラフを生成する際に同一の Σ 式をノード名として持つノードを一つのノードにまとめてダグ (有向無閉路グラフ) の形で表現した。これにより、同一のプレスブルガー文の真偽判定の重複を防ぐことが出来る。尚、提案する方法でエラーリカバリ性

を検証する場合、 $\forall z_1 \dots z_t (A)$ の形のプレスブルガー文が真であることを証明しなければならない。

$\forall z_1 \dots z_t (A)$ は、 $\text{not} \exists z_1 \dots z_t (\text{not} A)$ ($\text{not} A$ が整数解を持たないこと) と等価である。 $\text{not} A$ は積和形展開することによって " $A_1 \text{ or } A_2 \text{ or } \dots \text{ or } A_k$ "

(但し、各 A_i は幾つかの線形不等式の論理積) に展開できる。すべての A_i が整数解を持たなければ $\text{not} A$ も整数解を持たない。各 A_i が整数解を持たないことの証明は、 A_i の各線形不等式を整数線形計画問題の制約条件とみなして、その整数線形計画問題が整数解を持たないことを証明すればよい。この方法を用いて、 $\forall z_1 \dots z_t (A)$ が真であることを証明する。但し、 A_i が整数解を持たないことを証明するのは一般には時間がかかると思われるので、作成したプログラムでは、シンプレックス法を用いて A_i が実数解を持たないことを証明している。この方法では、整数解の非存在性を直接証明する代わりに、十分条件として、実数解の非存在性を証明しているが、実際に行ったいくつかの検証例ではすべてこの方法で証明が成功した。

また、検証者が与えた Φ に対して、 $M1$, $M2$ が Φ に対するエラーリカバリ性を持つと証明できなかった場合でも、どの状態対に対する Φ -到達可能性グラフが構成できなかったかや、その状態対に対する深さ k の到達可能性グラフの内容を表示する。これらの情報から、仕様記述の誤りや、 Φ に対するエラーリカバリ性の証明を行う際に検証者が補題としてどのような P 文 Q を指定すればよいか等に関する有益な情報が得られる。

検証例として、データリンクレベルにおける HDLC 手順 (正規応答モード) の一次局と二次局に相当する 2 つのプロトコルマシン $M1, M2$ に対して、「一次局の送

信状態変数と二次局の受信状態変数の値が一致している、あるいは、一次局が回線の永久障害を検出したことを表す特別な状態にいる」とき真であるようなP文Φを定め、作成したプログラムを用いて、M1, M2がΦに対するエラーリカバリ性を持つかどうかを調べた。これらのプロトコルマシンはあるメーカーの作成した装置を今のモデルで記述したものであり、入力記号は一次局が15個、二次局が12個である。また、プロトコルマシンの状態数は各々12個ずつ、レジスタの数は一次局6個、二次局5個である。

一般に、HDLC手順には幾つかのパラメータ（同一シーケンス番号のI（情報）フレームの最大再送回数、相手局が受信したことの確認を受けずに送信できる最大Iフレーム数（ウィンドウサイズ）、タイムアウトの発生回数の上限値）があり、これらの値は各製造者が独自に指定できるようになっている。これらのパラメータ値を大きくすると証明に要する時間も増大する。表2.1に幾つかのパラメータ値の組に対する検証結果を付す。尚、2.7では、以上述べた検証法を改良し、プロトコルが上述のようなパラメータ値をもつ場合に、パラメータ値に依存しない計算時間でエラーリカバリ性を機械的に検証する方法を述べる。

表2.1 命題2を用いた検証結果

	パラメータ値			
	1	2	3	4
Iフレームの最大再送回数	1	2	3	4
ウィンドウサイズ	1	2	3	4
タイムアウトの発生回数の上限	1	2	3	4
証明に要したCPU時間（分）	9	53	242	1372
グラフルガ-文判定回数（回）	8446	47952	254109	1280056
Φ-到達可能性の頂点の個数（個）	6172	34907	184749	930407
Φ-到達可能性のサイズの最大値	18	23	28	33

2.7 パラメータ値に依存しない検証法

図2.1のM1の初期状態 S_1 と図2.2のM2の初期状態 t_1 とからなる全状態において、通信系の動作 $[D, *]$ が実行されたとする（図2.4参照）。このとき、 M_1, M_2 の状態は、それぞれ、 S_2, t_1 となり、M1のレジスタ X_2 の値には1が加えられる。この後、 $[T, *]$ と $[D, *]$ がこの順に2回ずつ実行された後、さらに $[T, *]$ が実行されると、通信系はΦを満たす全状態に遷移する（図2.4）。さて、図2.1において、条件式 $\text{not}(x_1 \geq 2)$ をすべて $\text{not}(x_1 \geq 100)$ で置き換えた通信系 CS_{100} を考える。CS₁₀₀では、 $[T, *]$ と $[D, *]$ がこの順に100回ずつ実行された後、さらに $[T, *]$ が実行されて、初めてΦを満たす全状態に遷移する。このように、パラメータ値“2”を“100”に変更したことにより、Φを満たす全状態に遷移するのに必要な $[T, *]$ と $[D, *]$ の実

行回数が増加するので、 Φ -到達可能性グラフのサイズも図2.4に比較して非常に大きくなり、従って、検証に要する計算時間も膨大となる。

以下では、2.6で述べた検証手続きをこのようなパラメータ値に依存しない計算時間で実行できるように改良する。まず、 Φ -到達可能性グラフに関する若干の定義を与える。 Φ -到達可能性グラフにおいて、あるノードNとその子孫であるノードN'について、NとN'の第1成分と第2成分同志が等しい（すなわち、NとN'の表す二つの全状態において、対応するプロトコルマシンの有限制御部の状態がそれぞれ等しい）と仮定する。このとき、NからN'へ至る道を、「疑似閉路」と呼び、ノードNをその疑似閉路の始点、N'を終点と呼ぶ。もし疑似閉路Cが他の疑似閉路を含まないとき、Cを「単純疑似閉路」と呼ぶ。例えば、CS100の Φ -到達可能性グラフにおいて、

$$\begin{aligned} & \langle s_2, t_1, \langle x_1, x_2+1, x_3 \rangle, \langle y_1 \rangle \rangle \\ & \rightarrow \langle s_3, t_1, \langle x_1+1, x_2+1, x_3 \rangle, \langle y_1 \rangle \rangle \\ & \rightarrow \langle s_2, t_1, \langle x_1+1, x_2+1, x_3+1 \rangle, \langle y_1 \rangle \rangle \quad (7-1) \end{aligned}$$

は単純疑似閉路である。 Φ -到達可能性グラフに単純疑似閉路が存在するとき、C上の有向辺のラベルによって表される通信系の動作の系列が繰り返し実行される可能性がある。そのような動作系列を、「単純疑似閉路Cに対応する（通信系の）動作系列」と呼ぶ。例えば、動作系列[T,*],[D,*]は、上述の単純疑似閉路(7-1)に対応する動作系列である。

eを Φ -到達可能性グラフの有向辺とする。eのラベルの表す通信系の動作が、各レジスタの値の内容を定数（0を含む）分だけ増加または減少させる動作であるとき、eを「定数更新辺」と呼ぶ。ある単純疑似閉路C

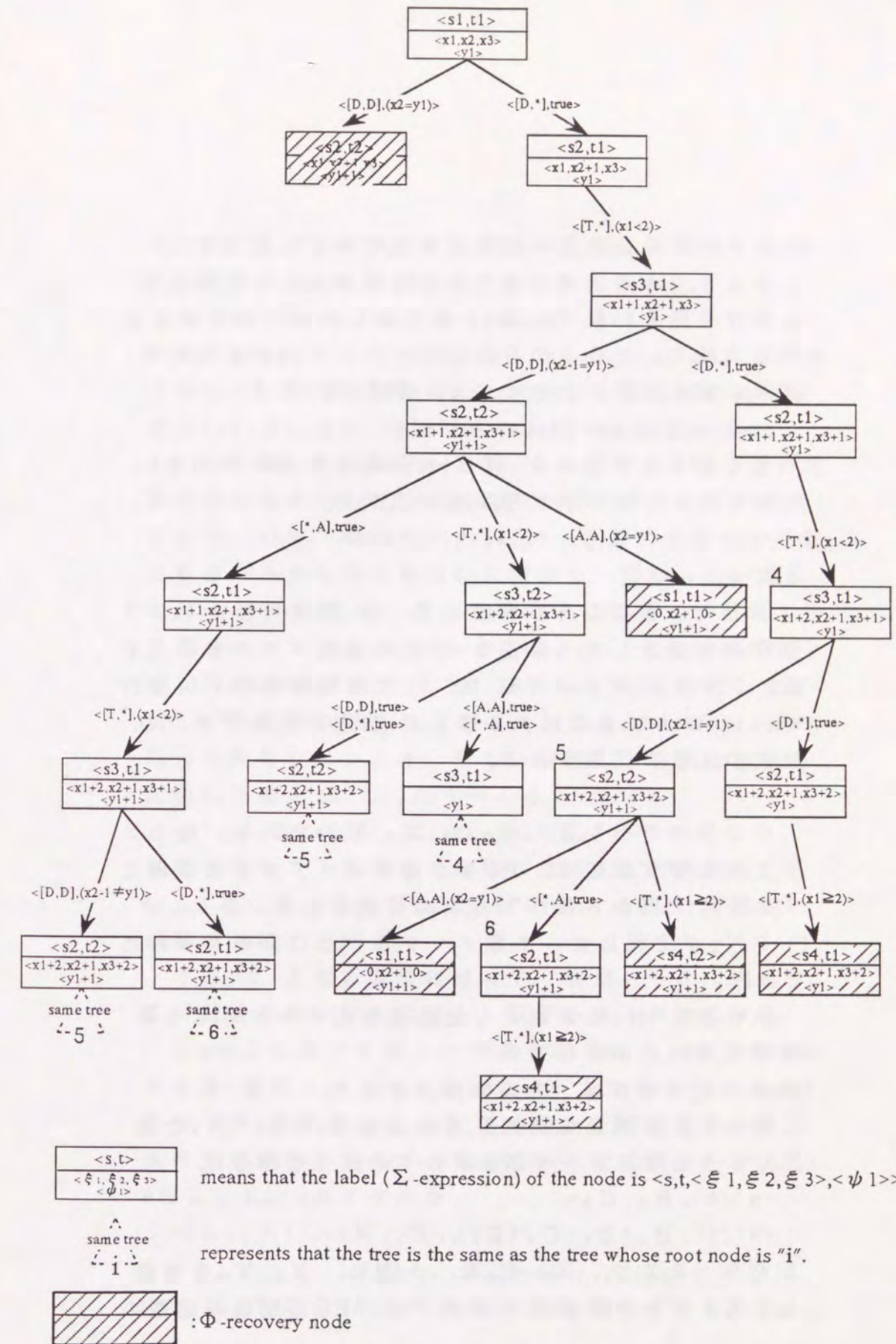


図2.4 M1, M2の Φ -到達可能性グラフ

上のすべての有向辺が定数更新辺であると仮定する。
このとき、ある Σ 式で表される全状態からCに対応する動作系列を k_1 回 ($k_1 \geq 0$) 繰り返した後の全状態を表す Σ 式は、 k_1 とC上の有向辺のラベルの内容を用いて、容易に書き表すことができる。例えば、 Σ 式

$$\langle s_2, t_1, \langle x_1, x_2+1, x_3 \rangle, \langle y_1 \rangle \rangle$$

で表される全状態から、(7-1)に対応する動作系列を k_1 回繰り返した後の全状態を表す Σ 式は、

$$\langle s_2, t_1, \langle x_1+k_1, x_2+1, x_3+k_1 \rangle, \langle y_1 \rangle \rangle$$

となる。

以上のようなことを考慮して、 Φ -到達可能性グラフの定義を変更して、修正 Φ -到達可能性グラフを導入する。プロトコルマシンM1, M2, (有限制御部の) 状態対 $\langle s_0, t_0 \rangle$ および Φ に対する修正 Φ -到達可能性グラフGは次のように定義される。

Gの根のラベルは、 $\langle s_0, t_0, X_n, Y_m \rangle$ である。他のノードおよび有向辺は、2.5の遷移グラフの定義を満たすように“根から葉へ”と順に定義される。但し、ノード N_2 が定義に従って葉ノードとしてGに導入されたときに、

条件SCC “ N_2 を終点とし定数更新辺のみからなる単純疑似閉路Cが存在する”

が成り立つならば、次の操作Pを行う。

〔操作P〕単純疑似閉路Cの始点を N_1 とし、 N_1 と追加しようとするノード N_2 のラベルを、それぞれ、

$$\langle s', t', B_n, C_m \rangle$$

$$\langle s', t', B_n+D_n, C_m+E_m \rangle$$

とする。ここで、 B_n, C_m は、一般に、 X_n, Y_m を変数とするレジスタ計算式であり、 D_n, E_m は整数の定数の

n 字組、 m 字組である。このとき、新しい変数 k_1 と適当な整数の定数 h_1 を導入し、 N_2 のラベルを、

$$\langle s', t', B_n+k_1 D_n, C_m+k_1 E_m \rangle [h_1]$$

に置き換える。このラベルは、ノード N_1 が表す全状態において、単純疑似閉路Cに対応する通信系の動作系列が k_1 ($1 \leq k_1 \leq h_1$) 回実行された後の全状態を表す。

一般には、根からノード N_1 への道の上には、既に定数更新辺のみからなる単純疑似閉路が存在する可能性がある。いま、根からノード N_1 への道に含まれる定数更新辺のみからなる単純疑似閉路を、根から近い順に $C_1, \dots, C_{q-1}$ とし、それらに対応して導入した新しい変数を $k_1, \dots, k_{q-1}$ とする。このとき、上述の条件がノード N_2 について成り立ち、しかも、 N_2 を終点とする新しい単純疑似閉路が、 $C_1, \dots, C_{q-1}$ のいずれとも有向辺を共有しないとき、新しい変数 k_q と適当な整数の定数 h_q を導入し、 N_2 のラベルを、

$$\langle s', t', B_n+k_q D_n, C_m+k_q E_m \rangle [h_q]$$

に置き換える。尚、以上の操作で導入した整数の定数 $h_1, \dots, h_q$ を、それぞれ、閉路 $C_1, \dots, C_q$ の「繰返し回数の上限」と呼ぶ。□

さらに、上述の変更に従って、復帰ノード、遷移する可能性のない枝の定義を、以下のように拡張する。

N_2 をGにおける葉ノードとする(但し、以下で定義する Φ -復帰ノードでも繰返しノードでもないとする)。Eを、 N_2 を始点とし、2.5で述べた〔遷移グラフの条件〕を満たすすべての有向辺からなる集合とする。 $e \in E$ とし、 e のラベルを

$$\langle [\alpha 1, \beta 1], c(K_q, X_n, Y_m) \rangle$$

(但し、 $K_q = \langle k_1, \dots, k_q \rangle$ とする)

とする。有向辺 e が次の条件(7-2)を満たすとき、 e は(

繰返し回数の上限 h_q について) 「遷移する可能性のない枝」と呼ばれ、そうでないとき、「遷移する可能性のある枝」と呼ばれる。

$\forall K_q, X_n, Y_m$

$$[\Phi(<s_0, t_0, X_n, Y_m>) \text{ and } (1 \leq k_1 + \dots + k_{q-1} + k_q \leq h_q) \\ \supset \text{not}(c(K_q, X_n, Y_m)))] \quad (7-2)$$

E に属する遷移する可能性のある枝は、すべてグラフに付け加えられる。ここで、頂点 N_2 のラベルに記述されている繰返し回数の上限 h_q は、それらの有向辺の終点にも書き込む。

このようにして付け加えられた有向辺の任意の終点を N_3 とする。但し、 N_3 のラベルを、

$$<s, t, B_n, C_m>[h_q]$$

(B_n, C_m は、 K_q, X_n, Y_m を変数とするレジスタ計算式)

とする。このとき、次の条件(7-3)が成り立つならば、 N_3 を Φ -復帰ノードと呼ぶ。

$\forall K_q, X_n, Y_m$

$$[\Phi(<s_0, t_0, X_n, Y_m>) \text{ and } (1 \leq k_1 + \dots + k_{q-1} + k_q \leq h_q) \\ \supset \Phi(<s, t, B_n, C_m>)] \quad (7-3)$$

Φ -復帰ノードには、以降それを始点とする有向辺は付け加えない。

G の根からノード N_2 への道に存在する単純疑似閉路を、根に近い順に、 $C_1, \dots, C_q$ とする。G に有向辺 $e: N_2 \rightarrow N_3$ 付け加えることにより、G に N_3 を終点とする単純疑似閉路 C_{q+1} が作られたと仮定する。次の(1)-(3)に場合分けして考える。

(1) C_{q+1} に定数更新辺でない有向辺が含まれているとき、この手続きは停止する。このとき、与えられた $M1, M2, <s_0, t_0>$ と Φ に対して、修正 Φ -到達可能性グラフ

は定義されない。

(2) C_{q+1} に含まれる有向辺がすべて定数更新辺であり、 C_{q+1} に対応する通信系の動作系列が、ある $C_i (1 \leq i \leq q)$ に対応する通信系の動作系列と一致するとき、ノード N_3 を、「繰返しノード」と呼ぶ。 Φ -復帰ノードの場合と同様、繰返しノードには、それを始点とする有向辺は付け加えない。

(3) C_{q+1} に含まれる有向辺がすべて定数更新辺であり、 C_{q+1} に対応する通信系の動作系列が、どの $C_i (1 \leq i \leq q)$ に対応する通信系の動作系列とも一致しないとき、 N_3 のラベルを、前述の[操作 P]で述べたラベルに置き換える。

このようにして G に付け加えたノード N_3 が、 Φ -復帰ノードでも繰返しノードでもない場合、以上の手続きを繰り返す。

上述の手続きを実行して、もうそれ以上付け加えるノードがなくなったならば、この手続きは停止する。このときに得られた G が、次の条件(A),(B)をともに満たすならば、G を、与えられた $M1, M2, <s_0, t_0>$ と Φ に対する修正 Φ -到達可能性グラフであると定義し、そうでなければ、修正 Φ -到達可能性グラフは定義されない。

(A) G のすべての葉ノードは、繰返しノードかまたは Φ -復帰ノードである。

(B) G における任意の単純疑似閉路 C について、C に次の条件(7-4)を満たす有向辺 $u \rightarrow v$ が含まれる。

$\forall K_q, X_n, Y_m$

$$[\Phi(<s_0, t_0, X_n, Y_m>) \text{ and } (h_q \leq k_1 + \dots + k_{q-1} + k_q) \\ \supset \text{not}(c(K_q, X_n, Y_m)))] \quad (7-4)$$

但し、有向辺 $u \rightarrow v$ のラベルを、

$\langle [\alpha 1, \beta 1], c(K_a, X_n, Y_n) \rangle$

とし, h_a を, ノード u に記述されている繰返し回数の上
限とする.

いま, $M1$ と $M2$ からなる通信系が全状態 $\langle s_a, t_a, R_n, V_n \rangle$ にあるとし, $\langle s_a, t_a, R_n, V_n \rangle$ が Φ を満たすと仮定する. もし, $\langle s_a, t_a \rangle$ に対して, 修正 Φ -到達可能性グラフが存在するならば, 通信系は, いつかは Φ を満足する全状態に遷移する. 従って, 2.5の命題2の手続きにおいて Φ -到達可能性グラフを修正 Φ -到達可能性グラフに置き換えて得られる手続きは, Φ に対するエラーリカバリ性の検証に用いることができる. この手続きの計算時間は, 本節冒頭で述べたパラメータ値に依存しない. さらに, 修正 Φ -到達可能性グラフの定義からこの手続きはいつかは必ず停止するので, グラフのサイズの上限 k を指定する必要はない. 但し, 定数更新辺のみからなる各単純疑似閉路 $C_i (i=1, 2, \dots)$ について, 繰返し回数の上限 $h_1, h_2, \dots$ を検証者が与える必要がある.

図2.4に対応する修正 Φ -到達可能性グラフを図2.5に示す. 前述の単純疑似閉路(7-1)は, 定数更新辺のみからなるので, その終点のラベル

$\langle s_2, t_1, \langle x_1+1, x_2+1, x_3+1 \rangle, \langle y_1 \rangle \rangle$ は,

$\langle s_2, t_1, \langle x_1+k_1, x_2+1, x_3+k_1 \rangle, \langle y_1 \rangle \rangle [2]$

に置き換えられる. 但し, 検証者によって, 繰返し回数の上限は2と指定されている. さらに, このノードを始点とする次のような単純疑似閉路が構成される.

$\langle s_2, t_1, \langle x_1+k_1, x_2+1, x_3+k_1 \rangle, \langle y_1 \rangle \rangle$
 $\rightarrow \langle s_3, t_1, \langle x_1+k_1+1, x_2+1, x_3+k_1 \rangle, \langle y_1 \rangle \rangle$
 $\rightarrow \langle s_2, t_1, \langle x_1+(k_1+1), x_2+1, x_3+(k_1+1) \rangle, \langle y_1 \rangle \rangle$
 (7-5)

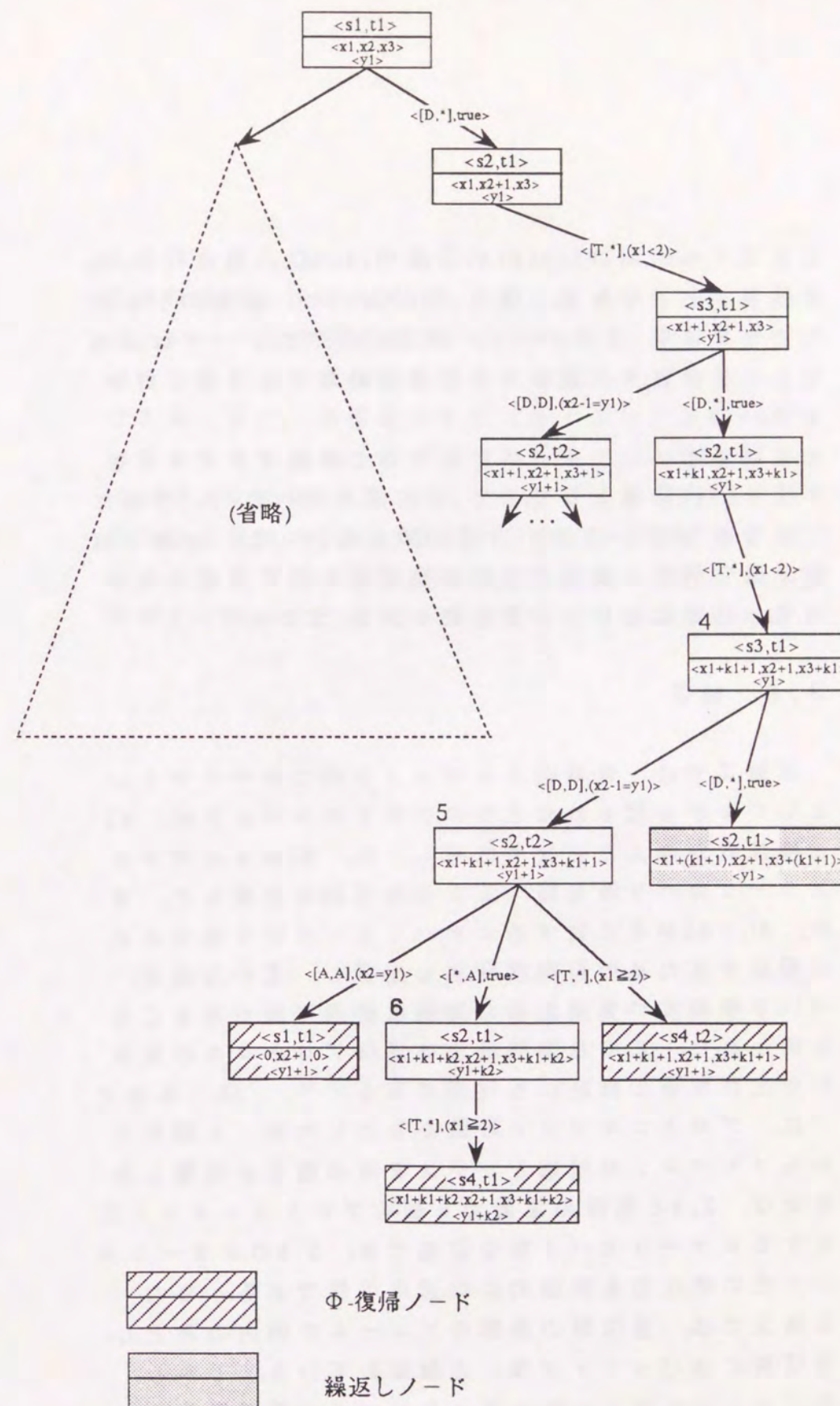


図2.5 修正 Φ -到達可能性グラフ

ここで、 $\langle s_2, t_1, \langle x_1 + (k_1 + 1), x_2 + 1, x_3 + (k_1 + 1) \rangle, \langle y_1 \rangle \rangle$ は、繰返しノードである。図2.5において、斜線の引かれたノードは Φ -復帰ノード、網掛けされたノードは繰返しノードを表す。遷移する可能性のない枝は書いていない。

本節で述べたアイデアに基づいて検証プログラムを作成し、検証例として、2.6末尾で述べたHDLC手順について Φ -エラーリカバリ性の検証を行った。命題2の検証法を用いた場合と本節の検証法を用いた場合について、検証に要した計算時間を表2.2に示す。

2.8 結言

本論文では、有限個のレジスタを持つオートマトンとしてモデル化された2つのプロトコルマシンM1, M2と検証者が与えたP文 Φ に対し、M1, M2が Φ に対するエラーリカバリ性を持つことを形式的に定義した。また、M1, M2が Φ に対するエラーリカバリ性を持つことを保証するための自動検証法を提案し、その方法が、HDLC手順程度の実用規模の問題に適用可能であることを確かめた。この自動検証法は通信プロトコルの安全性や生存性等の検証にも利用できる⁽¹³⁾。尚、本論文では、プロトコルマシンの数を2としたが、n個のプロトコルマシンの状態とレジスタ値の組を全状態とみなせば、2.4と同様の定義でn個のプロトコルマシンに対するエラーリカバリ性を定義でき、2.5のエラーリカバリ性の検証法も原理的には適用可能である。また、本論文では、通信路の異常をフレームの消失のみとし、通信路にはバッファがないと仮定している。しかし、各プロトコルマシンにエラーフレームの受信動作とい

う動作を導入し、M1の送信動作とM2のエラーフレームの受信動作を同時に実行することによって通信路上でのフレームの変形を表すとみなせば、フレームの変形が起こるような通信路に対しても提案する方法が適用できる。また、通信路が有界(サイズk)のFIFOキューとみなせる場合、各プロトコルマシンに新たにk個のレジスタを導入し、それらのレジスタでFIFOキューの内容を保持すれば、提案する検証法が適用できる。

尚、本章で述べた検証法が実用規模の問題にどの程度適用可能かどうかや、その評価が今後の課題である。

表 2. 2 検証に要した計算時間の比較

(a) 命題 2 を用いた検証結果

	パラメータ値			
I フレームの最大再送回数	1	2	3	4
ウィンドウサイズ	1	2	3	4
タイムアウトの発生回数の上限	1	2	3	4
証明に要した CPU 時間 (分)	9	53	242	1372
テストプログラム-文判定回数 (回)	8446	47952	254109	1280056
Φ -到達可能性グラフの頂点の個数 (個)	6172	34907	184749	930407
Φ -到達可能性グラフのサイズ の最大値	18	23	28	33

(b) 2.7 の方法を用いた検証結果

	パラメータ値
I フレームの最大再送回数	8
ウィンドウサイズ	7
タイムアウトの発生回数の上限	8
証明に要した CPU 時間 (分)	270
テストプログラム-文判定回数 (回)	96590
Φ -到達可能性グラフの頂点の個数 (個)	73776
Φ -到達可能性グラフのサイズ の最大値	29

第 3 章

通信プロトコルの代数的
仕様からプログラムへの変換

3.1 序言

代数的仕様記述法⁽¹⁷⁾は、厳密な意味の定義が簡明であり、検証の際に必要な仕様の無矛盾性や詳細化等の概念も、合同関係という概念のみを用いて簡明に定義できる。代数的仕様の部分クラスとして、抽象的順序機械の形で記述された仕様⁽¹⁸⁾、⁽¹⁹⁾がある。この仕様では、論理型、整数型等の基本タイプを前提として、状態の構造を陽にはもたない抽象的状态を表すタイプを定義する。関数には状態遷移関数、状態成分関数（抽象的状态からその成分の値を取り出す関数）、および補助関数があり、各状態遷移後の状態成分関数の値を公理によって定義する。抽象的状态は構造を陽にはもたないことから、新たに状態成分を追加する際、既に記述した部分を変更することなく、状態成分関数に関する構文と公理を局所的に追加するだけでよい、という特長がある。また、既にHDL、OSIセッション層等の通信プロトコルやスクリーンエディタの仕様を抽象的順序機械の形で記述し、仕様の形式的検証が行われ、本記述法の有効性が実証されている⁽¹⁹⁾⁻⁽²¹⁾。さらに、自然語による要求定義から抽象的順序機械の形で記述された仕様を導出する方法論も検討されている⁽²²⁾。

本章では、抽象的順序機械の形で記述された通信プロトコルの代数的仕様からプログラムへの変換について述べる。まず3.2で、関連する諸定義と仕様の例を示す。3.3では、文献(2)、(3)の方針に従って、与えられた仕様から効率の良い手続き型プログラムへの変換法を述べ、目的プログラムを解釈実行する機械を定義する代数的仕様、および変換を定義する代数的仕様を示

す。さらに3.4では、この変換法に従って、与えられた仕様をCプログラムに変換するコンパイラについて述べる。

3.2 抽象的順序機械

3.2.1 代数的仕様

仕様記述には、言語ASL/*⁽²³⁾を用いる。ASL/*では、始記号を指定しない文脈自由文法Gと公理の集合AXの組 $t=(G, AX)$ を仕様と呼ぶ。Gの非終端記号Aから生成される終端記号列全体からなる集合を $L_G(A)$ 、Gのいずれかの非終端記号から生成される終端記号列全体の集合を E_t と書き、 E_t の元をtの表現式と呼ぶ。タイプ名と、そのタイプの元を表す表現式を生成する非終端記号を同一視して、 $L_G(A)$ に属する表現式を、タイプAの表現式と呼ぶ。また、生成規則 $A \rightarrow f(A_1, A_2, \dots, A_n)$ (A, $A_1, A_2, \dots, A_n$ は非終端記号、f、括弧“(”, “)”およびコンマ“, ”は終端記号)が記述されているとき、終端記号fを関数名とみなし、関数fの引数のタイプは $A_1, A_2, \dots, A_n$ であり、関数値のタイプはAであるという。

論理型のtrue, false, 任意の非負整数の10進表現0, 1, 2, ...等のように、それ自身が値を表すと考えられる表現式を構成子表現式と呼ぶ。構成子表現式の集合は、それを生成する非終端記号の部分集合Vcを指定することにより、 $\bigcup_{A \in Vc} L_G(A)$ と定義する。

公理とは変数を含む表現式の対 $Q == r$ である。但し、公理に現れる各変数xにはGの非終端記号Mxが一つ割り当てられているとし、変数xのタイプはMxであるという。公理 $Q == r$ は、Qまたはrに現れる各変数に、そのタイ

プである非終端記号から生成される任意の表現式を代入して得られる終端記号列の対が、仕様 t において合同となることを表す。形式的には、 AX のすべての公理を満たす E_t 上の最小の合同関係⁽²³⁾を仕様 t の指定する合同関係といい、 $\equiv_t$ で表す。 $e \equiv_t e'$ が成り立つとき、表現式 e と e' は仕様 t において合同であるという。

3.2.2 抽象的順序機械の定義

仕様 t が次の条件(1)~(3)をすべて満たすとき、 t を抽象的順序機械の形で記述された仕様と呼ぶ。

(1) t の文法は、抽象的状态を表す表現式を生成する非終端記号 (state とする) をもつ。

(2) t の関数は、(2a) 状態遷移関数 (関数値のタイプは state) ,

(2b) 状態成分関数 (第1引数のタイプは state, 関数値のタイプは基本タイプ) , および (2c) 補助関数 (関数値のタイプは基本タイプ) に分類される。これらはいずれも、タイプが state である引数を二つ以上もってはいない。簡単のため、タイプが state の引数をもつとき、それは第1引数であるとする。状態成分関数 0 は、タイプが state の引数が表す抽象的状态において、 0 の表す特定の状态成分 (そのタイプは、例えば整数や配列等、状態成分ごとに定められた任意の基本タイプ) の値を表す。以下 (2a), (2b) をそれぞれ単に遷移関数、成分関数と呼ぶ。

(3) t の公理は次の条件(3a)~(3c)をすべて満たす。

(3a) 公理の左辺は線形で、かつ重なりをもたない⁽²⁴⁾。

(3b) 公理の右辺に現れる変数は、その左辺にも現れる。

(3c) 公理は次の(i)~(iii)のいずれかの形をしている。

これらを順に i~iii 型公理と呼ぶ。

(i) $0(T(x_1, x_2, \dots, x_n), u_2, \dots, u_m) == \dots$

(ii) $T'(x_1, x_2, \dots, x_n) == \dots$

(iii) $A(x_1, x_2, \dots, x_n) == \dots$

但し、 T, T' は遷移関数、 0 は成分関数、 A は補助関数、 $x_1, x_2, \dots, x_n$ は変数、 $u_2, \dots, u_m$ は変数か構成子表現式である。i, iii 型公理の右辺に遷移関数が現れてはならない。i, ii 型公理の左辺に現れる遷移関数 (条件(3a)より i, ii 型両方の公理の左辺に現れることはない) を、それぞれ I, II 型の (状態) 遷移関数と呼ぶ。□

条件(3c)より、i, iii 型公理の右辺に遷移関数は現れないので、状態遷移後の成分関数の値は、遷移前の状態における成分関数と補助関数の値と、遷移関数の state 以外のタイプの引数のみに依存して定まる。

抽象的順序機械の形で記述された仕様の例として、OSI セッションプロトコル⁽²⁵⁾を取り上げる。セッション層の通信主体 SPM (セッションプロトコルマシン) の仕様 t_{SPM} ^{(28), (29)} の一部を表 3.1 に示す。但し、表現式の構文を定める文法は省略している (以下でも同様)。MAP(s, spdu) は、SPM が状態 s において相手 SPM の送信した MAP という種類のデータ (spdu はその内容を表す) を受信したときの動作を表す II 型の遷移関数であり、[1] は、MAP(s, spdu) の値が、そのときの SPM のフェーズ (成分関数 phase(s) で表される) に従って定まる I 型の遷移関数の値と合同となることを表す ii 型公理である。[2]~[6] は、I 型の遷移関数 MAP1 による遷移後の成分関数の値を定義した i 型公理である。例えば [3] は、遷移 MAP1 後の V[M] (SPM の一つの内部変数の内容を表す成分関数) の値が遷移前の値に 1 加えたものに等しいことを表す。[7], [8] は補助関数 p12, p19 を定義する iii 型公理である。

表 3.1 SPMの仕様 t_{SPM} (部分)

```

MAP(s, spdu) ==
  if phase(s) = idle & TCcon then err1(s)
  else if phase(s) = data_trans then
    if  $\neg SSYNM(s) \wedge p12(s) \wedge p19(s, spdu)$ 
    then MAP1(s, spdu) else err0(s)
  else if phase(s) = abort then s
  else err0(s) [1]
V[A](MAP1(s, spdu))
  == if Vsc(s) then V[A](s) else V[M](s) [2]
V[M](MAP1(s, spdu)) == V[M](s) + 1 [3]
SSYNM(MAP1(s, spdu)) == true [4]
phase(MAP1(s, spdu)) == phase(s) [5]
OWNED(MAP1(s, spdu), x) == OWNED(s, x) [6]
p12(s) == Ac(s, dk)  $\wedge$  Ac(s, mi)  $\wedge$  AAc(s, ma) [7]
p19(s, spdu) == sn(spdu) = V[M](s) [8]
...

```

3.3 プログラムへの変換法

ここでは、抽象的順序機械の形で記述された仕様からプログラムへの変換法を述べる。変換法を形式的に記述するため、以下の(a)~(d)の仕様を考える。

- (a) 抽象的順序機械の形で記述された入力仕様 t_{ASM} .
- (b) 目的プログラムを解釈実行する機械 OM を定義する仕様 t_{OM} (3.3.1 参照).
- (c) 入力仕様 t_{ASM} に対する目的プログラムを定義する仕様 t_{COMP} (3.3.2.1 参照). t_{COMP} は, t_{ASM} を引数としてそれを実現する目的プログラムを値とする関数を定義するため、仕様 t_{ASM} やその中の関数名等を表現式として扱う。以下では表現式としての仕様は $[t_{ASM}]$ と書き、仕様そのものと区別する。関数名についても同様である。従って厳密には t_{COMP} において入力仕様 $[t_{ASM}]$ の構

文を定義しなければならないが、以下では省略している。

(d) t_{ASM} の各関数を t_{OM} と t_{COMP} の各関数によってどのように実現するかを指定する仕様 $CRP(t_{ASM})$ (3.3.2.1 参照). $CRP(t_{ASM})$ は, t_{ASM} の関数名等に依存して定まる。 t_{ASM} に対して対応関係を指定する仕様 $CRP(t_{ASM})$ を与える写像 CRP を, 対応関係の指定と呼ぶ。

3.3.1 目的機械 OM

目的プログラムの構文を定義する生成規則 (これらは次に述べる仕様 $t_{OM} = (G_{OM}, AX_{OM})$ の文法 G_{OM} の生成規則の集合に含まれる) を表 3.2 に示す。プログラム変数は $V_1, V_2, \dots$ であり, プログラムは変数への代入文と IF 文の有限系列である。厳密には, 各プログラム変数はタイプをもち, それに従って正しい構文をもつプログラムが定義されるが, 簡単のため, その詳細は省略している。また, 表 3.2 の非終端記号 `program` から生成される表現式の部分系列に対して, それらの接続を表す関数が定義されており, 表 3.2 の生成規則の右辺において, 隣接する各終端記号や非終端記号の間には, 接続を表す関数が挿入されていると考えるが, 表では省略している。 t_{COMP} で扱う入力仕様の構文についても同様である。

OM は, 抽象的順序機械の形で記述された仕様 t_{OM} によって定義される。 t_{OM} は表 3.3 に示すような関数をもつ。OM の状態 s_{OM} におけるプログラム変数 V_j ($j \geq 1$) の値は, 同じ名前の成分関数 $V_j(s_{OM})$ で表される。 t_{OM} を表 3.4 に示す。公理 [11], [12] はそれぞれ, 代入文, IF 文から始

表 3.2 目的プログラムの構文

program	→ ε statement ';' program
statement	→ 'IF' expression 'THEN' program 'ELSE' program variable ':=' expression
expression	→ variable constant '¬' expression expression '+' expression ...
variable	→ 'V ₁ ' 'V ₂ ' ...
constant	→ 'TRUE' 'FALSE' '0' '1' ...

まるプログラムの実行を定義する。関数 EXEC は、本来は、

```
EXEC(som, prog')
== if HEAD(prog') =  $\varepsilon$  then som
   else HEAD(prog') = 'Vj := exp' then
       EXEC(Assign(som, Vj, exp), TAIL(prog'))
   ...
```

のように、ii 型公理で定義されている。ここで、HEAD(prog') および TAIL(prog') は、それぞれ、プログラム prog' の先頭の文（但し、prog' に含まれる文が 0 個なら ε ）および、prog' から HEAD(prog') を取り除いたプログラムを表す関数である。仕様の読みやすさのため、表 3.4 では、[10]~[12] のように左辺で場合分けをしている。他の関数も同様である。[13]~[14] は代入文の実行を定義している。[15]~[18] は、プログラム中の式の値を評価する関数 EVAL を、式の構造に基づいて再帰的に定義している。@ は基本タイプの演算子を表す。但し、OM における基本演算はすべて全域関数とする。

表 3.3 仕様 tom の関数

状態成分関数	
$V_1(som), V_2(som), \dots$	それぞれ、OM の状態 som におけるプログラム変数 $V_1, V_2, \dots$ の内容を表す。
状態遷移関数	
INIT_OM	OM の初期状態を表す。
EXEC(som, prog)	状態 som においてプログラム prog を実行した後の状態を表す。
Assign(som, V _j , exp)	状態 som において式 exp を評価した値を変数 V _j に代入した後の状態を表す。
補助関数	
EVAL(som, exp)	状態 som において式 exp を評価した値を表す。

表 3.4 目的機械 OM の仕様 tom

$V_i(\text{INIT_OM})$	== 0	[9]
EXEC(som, ε)	== som	[10]
EXEC(som, 'V _j := exp ';' prog)	== EXEC(Assign(som, j, exp), prog)	[11]
EXEC(som, 'IF' exp 'THEN' prog ₁ 'ELSE' prog ₂ ';' prog)	== if EVAL(som, exp) then EXEC(EXEC(som, prog ₁), prog) else EXEC(EXEC(som, prog ₂), prog)	[12]
$V_j(\text{Assign(som, V}_j, \text{exp)})$	== EVAL(som, exp)	[13]
$V_i(\text{Assign(som, V}_j, \text{exp)})$	== $V_i(\text{som})$ ($i \neq j$)	[14]
EVAL(som, '@' exp)	== @ EVAL(som, exp)	[15]
EVAL(som, exp ₁ '@' exp ₂)	== EVAL(som, exp ₁) @ EVAL(som, exp ₂)	[16]
EVAL(som, 'V _i ')	== $V_i(\text{som})$	[17]
EVAL(som, 'c')	== c	[18]

3.3.2 変換法の概要

3.3.2.1 変換の基本方針

簡単のため、入力仕様 t_{asm} は、以下の条件(1)~(3)をすべて満たす抽象的順序機械の形で記述された任意の代数的仕様とする。一般の場合については3.3.2.2で述べる。 t_{asm} で定義される抽象的順序機械(ASMと呼ぶ)の状態を表すタイプを $stateASM$ とする。

(1) t_{asm} の成分関数と遷移関数は、タイプが $stateASM$ 以外の引数をもたない。

(2) 公理はすべて i 型である。

(3) 公理の右辺において、 if 関数は、右辺全体または他の $then$ 部または $else$ 部全体にしか現れない。

文献(2),(3)の方針に従い、以下のように変換を行う。 0_j ($1 \leq j \leq p$) を t_{asm} の成分関数、 T_i ($1 \leq i \leq q$) を遷移関数とする。目的プログラムは、各遷移関数 T_i を実現する以下のようなプログラム $OBJ([t_{asm}], [T_i])$ の集合である。 $OBJ([t_{asm}], [T_i])$ は、プログラム変数として、成分関数 0_j の値を保持する変数 W_j と、退避のための変数 R_j を用いる。但し、 OM の変数名は $V_1, V_2, \dots$ なので(3.3.1参照)、各 j ($1 \leq j \leq p$) について、 W_j は V_{2j-1} 、 R_j は V_{2j} で表すと考え(厳密には、成分関数の関数値と同じタイプをもつプログラム変数を用いなければならない)、以降、 W_j 、 R_j を用いて説明する。

以上のように、 t_{asm} の関数 0_j を OM のプログラム変数 W_j で、また、関数 T_i を $OBJ([t_{asm}], [T_i])$ で実現することを指定するのが、仕様 $CRP(t_{asm})$ である(表3.5)。表では省略しているが、 $CRP(t_{asm})$ の生成規則は、 t_{asm} の生成規則すべてと、表3.5で参照している t_{om} 、 t_{com}

表3.5 対応関係を指定する仕様 $CRP(t_{asm})$

$$\begin{aligned} 0_j(\text{map}(s_{om})) &== W_j(s_{om}) & [19] \\ T_k &== \text{map}(\text{EXEC}(\text{INIT_OM}, \text{OBJ}([t_{asm}], [T_k]))) & [20] \\ T_i(\text{map}(s_{om})) &== \text{map}(\text{EXEC}(s_{om}, \text{OBJ}([t_{asm}], [T_i]))) & [21] \end{aligned}$$

p の関数に関する生成規則とからなる。 $CRP(t_{asm})$ では、この対応関係を指定するために、関数 map を導入する。 $\text{map}(s_{om})$ は、 OM の状態 s_{om} が実現している ASM の状態を表す。公理[19]は、“ OM の状態 s_{om} の実現している ASM の状態 $\text{map}(s_{om})$ における関数 0_j の値は、 s_{om} における関数 W_j の値(変数 W_j の値)と合同である”ことを表す。[21]は、1引数の(タイプが $stateASM$ の引数のみをもつ)遷移関数 T_i について、“状態 $\text{map}(s_{om})$ において状態遷移 T_i が起こった後の ASM の状態は、 s_{om} において $OBJ([t_{asm}], [T_i])$ を実行した後の OM の状態の実現している ASM の状態と合同である”ことを表している。引数をもたない遷移関数(初期状態を表す関数)についても同様である([20])。これを図示すると、図3.1のようになる。破線の矢印は、 t_{asm} の関数が OM や目的プログラムによってどのように実現されるかを表す。

変換の正しさは、 t_{comp} と t_{om} の関数を用いて t_{asm} の関数を実現できることを示すことにより保証されるが、 $CRP(t_{asm})$ は、このような関数間の対応関係を仕様で陽に記述したものである(この対応関係のもとで t_{asm} の関数が正しく実現されることの形式的な証明については、文献(29)の4.を参照されたい)。

各プログラム $OBJ([t_{asm}], [T_i])$ ($1 \leq i \leq q$) の内容は、

状態遷移 T_i 後の各成分関数 $O_j (1 \leq j \leq p)$ の値の変数 W_j への代入文の系列である。例えば、表3.6の仕様は表3.7に変換される。i型公理の右辺に遷移関数は現れないので(3.2.2の定義参照)、右辺に現れる成分関数は、状態 s_{ASM} での(遷移前の)値を表している。従って、公理の右辺からこのような代入文を容易に生成できる。但し、 $OBJ([t_{ASM}], [T_i])$ では、まず W_j の値を局所変数 R_j に退避し、代入文右辺では退避変数の方を参照するようにする。なぜならば、もし退避を行わないとすると、状態遷移前後の成分関数の値の区別がプログラムでは行えず、正しく計算が行えないからである。

以上の方法で t_{ASM} を実現する目的プログラムを定義する仕様 t_{COMP} を表3.8に示す。 t_{ASM} の遷移関数 T_i を実現する $OBJ([t_{ASM}], [T_i])$ は、公理[22]より、 $comp(S_{of_ax}([t_{ASM}], [T_i]), \#0([t_{ASM}]))$ である。ここで、 $S_{of_ax}([t_{ASM}], [T_i])$ は、 t_{ASM} において T_i が左辺に現れるi型公理の系列を表す表現式であり、また、 $\#0([t_{ASM}])$ は、 t_{ASM} の成分関数の個数を表す。これらの関数を定義する公理は表3.8では省略している。 $comp(s_{of_ax}, p)$ は、 $reserve(p)$ と $comp_{s_of_ax}(s_{of_ax})$ の接続からなるプログラムである([23])。ここで、 $reserve(p)$ は、各 $j (1 \leq j \leq p)$ について順に、 W_j の値を R_j に退避する代入文の系列であり([24],[25])、 $comp_{s_of_ax}(s_{of_ax})$ は、i型公理の系列 s_{of_ax} に属する各公理' $O_j(T_i(s_{ASM})) == r_{ij}$ 'あるいは' $O_j(T_i) == r_{ij}$ 'に対して、状態遷移 T_i 後の O_j の値 r_{ij} を退避変数 $R_k (1 \leq k \leq p)$ を用いて計算して W_j に代入する文の系列である([26]~[35])。[28]~[35]において、 $c_exp_ax, c_exp1_ax, c_exp2_ax$ は、 t_{ASM} の公理右辺に現れる(一般にifを含む)表現式を表す変数、 $exp_ax, exp1_ax, exp2_ax$ は、ifを

含まない表現式を表す変数である。 $comp_{c_exp}$ や $comp_{exp}$ も、表3.4と同様、仕様の読みやすさのため、引数の場合分けを左辺で行っている。 $comp_{ax}(ax)$ は公理 ax に対応する一つの文を、 $comp_{c_exp}(j, c_exp_ax)$ は c_exp_ax に対応するIF文を、 $comp_{exp}(exp_ax)$ は exp_ax に対応するプログラムの式を表す。

state map(s_{OM}) of ASM

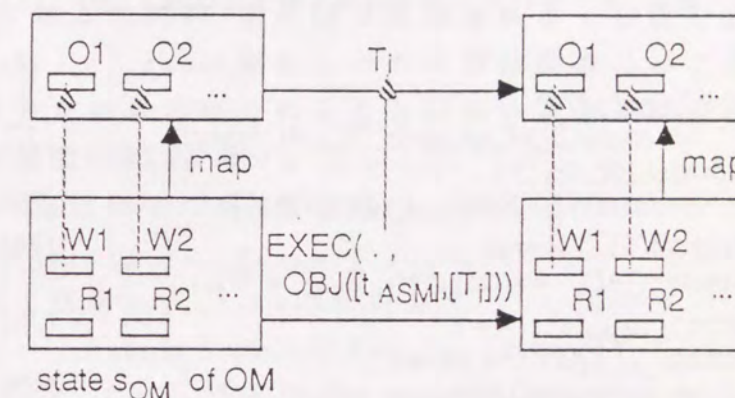


図3.1 OMと $OBJ([t_{ASM}], [T_i])$ によるASMの実現

表3.6 入力仕様 t_{ASM} の例

$$\begin{aligned} O_1(T_1(s)) &== O_1(s) + O_2(s); \\ O_2(T_1(s)) &== O_3(s); \\ O_3(T_1(s)) &== O_2(s) + O_3(s); \end{aligned}$$

表 3.7 表 3.6 を実現する目的プログラム

```

R1 := W1;
R2 := W2;
R3 := W3;

W1 := R1 + R2;
W2 := R3;
W3 := R2 + R3;

```

表 3.8 目的プログラムを定義する仕様 t comp

```

OBJ(spec, 'Ti')
== comp(S_of_ax(spec, 'Ti'), #O(spec)) [22]
comp(s_of_ax, j)
== reserve(j) comp_s_of_ax(s_of_ax) [23]
reserve(0) == ε [24]
reserve(j+1) == reserve(j) 'Rj+1 := Wj+1;' [25]
comp_s_of_ax(ε) == ε [26]
comp_s_of_ax(ax ';' s_of_ax)
== comp_ax(ax) comp_s_of_ax(s_of_ax) [27]
comp_ax('Oj(Ti(s_ASM)) == 'c_exp_ax'
== comp_c_exp(j, c_exp_ax) [28]
comp_ax('Oj(Ti) == 'c_exp_ax'
== comp_c_exp(j, c_exp_ax) [29]
comp_c_exp(j,
'if exp_ax then c_exp1_ax else c_exp2_ax' ==
'IF comp_exp(exp_ax) THEN comp_c_exp(j, c_exp1_ax)
ELSE comp_c_exp(j, c_exp2_ax)'; [30]
comp_c_exp(j, exp_ax)
== 'Wj := comp_exp(exp_ax)'; [31]
comp_exp('@' exp_ax) == '@' comp_exp(exp_ax) [32]
comp_exp(exp1_ax '@' exp2_ax)
== comp_exp(exp1_ax) '@' comp_exp(exp2_ax) [33]
comp_exp('Oj(s_ASM)') == 'Rj' [34]
comp_exp('c') == 'c' [35]

```

3.3.2.2 一般の場合の変換法

(a) i型公理 $0(T(x_1, x_2, \dots, x_n), u_2, \dots, u_m) = \dots$ において、遷移関数、成分関数の引数のうち、タイプが stateASM 以外のものの扱いを考える。

成分関数 0 の第 1 引数以外の引数のタイプを順に $A_2, \dots, A_m$ ($m \geq 2$) とし、これらのタイプの値（構成子表現式）が有限個しか存在しないと仮定する。0 は、タイプが $A_2, \dots, A_m$ である任意の構成子表現式の組 $\langle c_2, \dots, c_m \rangle$ に対して次の i 型公理で定義される 1 引数の成分関数 $0 \langle c_2, \dots, c_m \rangle$ の組を、一つの成分関数としてまとめて表すために用いられることが多い（例えば、表 3.1 の成分関数 OWNED 参照）。

“0 を左辺にもつ各 i 型公理

$0(T(x_1, x_2, \dots, x_n), u_2, \dots, u_m) = r,$

および各 $\langle c_2, \dots, c_m \rangle$ に対して、

$$0 \langle c_2, \dots, c_m \rangle (T(x_1, x_2, \dots, x_n)) = r \{u_2 / c_2, \dots, u_m / c_m\}$$

($\gamma \{ \alpha_1 / \beta_1, \dots, \alpha_n / \beta_n \}$ は、系列 γ の互いに重なり合わない部分系列 $\alpha_1, \dots, \alpha_n$ を、系列 $\beta_1, \dots, \beta_n$ に置き換えて γ から得られる系列を表す)

いま、第 i 引数 ($2 \leq i \leq m$) が k_i 個の値を取り得ると仮定する。プログラムでは、成分関数 0 の値を保持する変数として、 $i-1$ 番目の添字の範囲が $1 \sim k_i$ である $m-1$ 次元の配列を用い、各構成子表現式の組 $\langle c_2, \dots, c_m \rangle$ に対して配列の一つの要素を割り当て、 $0(s, c_2, \dots, c_m)$ の値をその要素内に保持する。

もし第 1 引数以外の引数のタイプが、値を無限個取り得るとすると、上のように添字の範囲が有限の配列を用いて実現することはできない。例えばリスト構造

を用いて公理による式の変換を直接行うことにより実現が可能ではあるが、実行効率は著しく低下する。

遷移関数 T の引数 $x_1, \dots, x_n$ については、 $OBJ([t_{asm}], [T_i])$ を、 $x_1, \dots, x_n$ に対応する引数をもつ手続きとして実現すればよい。

(b)ii, iii型公理は、左辺の関数の引数がすべて変数であり、いわゆる関数型プログラム⁽²⁶⁾の形をしているので、各公理に対応する手続きを導入し、公理右辺に現れる基本タイプの関数に対応する基本演算に、遷移関数、補助関数に対応する手続き呼出しに、成分関数を変数の参照にそれぞれ変換すればよい。

これらの拡張を可能にするため、3.3.1で述べた仕様 tom に、配列に関する基本演算と、手続き呼出しの機能を付け加え、手続きの引数によって t_{asm} の遷移関数や補助関数の $stateASM$ 以外の引数を実現することをCRPで指定する必要があるが、容易であるので省略する。

なお、変数 R_i への値の退避はすべての成分関数や補助関数の値に対して必要なわけではない。例えば、表3.7では変数 W_1, W_3 の値の退避は不要である。退避すべき値の個数が最小となる更新順を決める問題は、レジスタ割付けの問題として知られている⁽²⁷⁾。3.4で述べるコンパイラでは、待避する値の個数の“少ない”更新順を選ぶ（特に、値の待避が全く不要な更新順が存在する場合は、そのうちの一つが選ばれる）。

3.4 コンパイラの実現と変換例

3.4.1 プロトコルの代数的仕様のコンパイラ

3.3.2.2で述べた一般の場合の変換法に従い、C言語のプログラムを生成するコンパイラをUNIX上で実現した。以下では、その概要について述べる。

3.4.1.1 コンパイラへの入力

本コンパイラは、通信プロトコルの記述を行うことを考慮して、基本データタイプに①整数型②論理型③スカラ型④構造体型の4つを用意している。特に、スカラ型はトークン等の名前の集合、構造体型は送受信データ等を定義するのに有用である。

構造体型には、定数として、そのフィールドの値がすべて未定義の構造体を表す $null$ 、基本演算として、構造体 r のフィールド f の値を v に変更した構造体を表す $put(r, f, v)$ 、構造体 r のフィールド f の値を表す $get(r, f)$ がある。

3.4.1.2 基本データタイプの実現

整数型、論理型、スカラ型はC言語の int 型で実現する。 $define$ 文により論理型の定数 $true$ および $false$ にそれぞれ1と0、スカラ型を構成する各定数に異なる一つの整数を割り当てる。

構造体型はC言語の構造体を用いて実現する。また、構造体型の定数 $null$ を $define$ 文により0と定義する。

3.4.1.3 公理から代入文への変換

基本的には、3.3.2.2で述べた変換法に従って変換を行う。Cプログラムの代入文右辺では、公理右辺の状態成分関数に変数の値の参照に、整数型・論理型の関数がC言語の整数演算・論理演算に、関数ifがif文に相当する。また、関数putは構造体のフィールドへの代入文に、関数getは構造体のフィールドの参照に相当する。但し、putの引数がnullの場合、もし必要ならば、値の代入の前に関数mallocにより構造体の領域を確保する。

3.4.1.4 状態成分関数の値の更新順序

状態遷移に伴う状態成分関数の値の更新の際に、どの状態成分関数から値を更新するかによって、待避すべき値の個数が異なる場合がある。生成されるプログラムの効率を考えると待避する値の個数が少なくなるように、更新の順番を決めることが望ましい。ここでは、状態成分関数の値の更新順を決定するために、本コンパイラが用いている方法の概要を述べる。以下では、コンパイラの入力となる仕様に補助関数は存在しないとする。以下の議論はそれが存在する場合に容易に拡張できる。

現有コンパイラでは、各状態遷移関数 T_i に対して、次の条件(1),(2)をともに満たす有向グラフ G_i を作成する。 G_i は状態遷移 T_i の際の状態成分関数間の依存関係を表している。

(1)状態成分関数の集合と有向グラフ G_i の頂点の集合の間には1対1の対応がついており、有向グラフ G_i の各

頂点是对应する状態成分関数名をラベルとしてもつ。
(2)状態遷移 T_i 後の状態成分関数 0_j の値を定義するi型の公理の右辺に状態成分関数 $0_j'$ (但し $j \neq j'$)が現れているときかつそのときのみ、 0_j をラベルにもつ頂点から $0_j'$ をラベルにもつ頂点への有向枝が存在する。

有向グラフ G_i において、入射枝をもつ頂点に対応する状態成分関数 $0_j'$ はその有向枝を出射枝としてもつ頂点に対応する状態成分関数 0_j の値の定義に用いられているので、 $0_j'$ の更新は 0_j の更新より後に行うかまたは $0_j'$ の値を待避するかしなければならない。コンパイラでは、上記の性質を利用して、以下のように更新順を決める。

(a) まず、入射枝が一つもない頂点を任意に1つ選び、対応する関数の値を更新する代入文を生成する。また、その頂点とその頂点から出ている枝をすべて取り除く。以降、全ての頂点を取り除かれるか、入射枝を持たない頂点なくなるまでこれを繰り返す。もし、全ての頂点を取り除かれれば終了。さもなければ次の(b)を行う。

(b) どの頂点も少なくとも1つ入射枝を持つならば、適当に頂点を選び(本コンパイラでは出射枝の数が最も多い頂点を選ぶことにしている)、その関数について待避変数を作る。以後、この関数の値の参照は待避変数の方で行う。そして、その関数の値の更新を行う代入文を生成し、その頂点とその出射枝、入射枝をすべて取り除く。以上を行った後、(a)を実行する。

3.4.2 変換例

3.2.2で述べた仕様 t_{SPM} を入力例として、本コンパイラの生成する目的プログラムの効率等について述べる。

図3.2に、コンパイラへの実際の入力仕様のうち、I型の変移関数MAPとI型の変移関数MAP1に関する部分を示す(表3.1に対応)。図において、for each " α " in {" β_1 ", ..., " β_m " } " γ "は、 γ に現れる α を β_i ($1 \leq i \leq m$) に置き換えて得られる系列 $\gamma\{\alpha/\beta_1\} \dots \gamma\{\alpha/\beta_m\}$ を表すマクロ記法である。コンパイラによって生成されたCプログラムのうち、MAPおよびMAP1に対応する部分を、それぞれ図3.3の(a),(b)に示す。送受信データを表すタイプは、構造体型として定義されており、これらを表す成分関数(SSP1~SSP4, TSP, SPDU1~SPDU5)の値は、構造体型の変数に代入される。

図3.3(b)で、(1)状態遷移の前後で値が変化しない成分関数(phaseやOWNED)については不要な代入文は生成されておらず、また、(2)成分関数の値の待避が全く不要な値の更新順(3.3の末尾参照)のうちの一つが選択され、対応するプログラムが生成されている。このような最適化により、本例題では、生成された目的プログラムは、人手で作成した手続き型プログラムとほぼ同程度の効率で実行が可能である。

```
MAP(s,pa_MAP) ==
  if phase(s) = idle_and_TCcon then err1(s)
  else if phase(s) = data_trans then
    if not(SSYNM(s)) and pl2(s) and
      pl9(s,get(pa_MAP,sn))
      then MAP1(s,pa_MAP)
    else err0(s)
  else if phase(s) = abort then s
  else err0(s);

for each "X" in {"phase" "Vsc" "Vtca" "lc"}
  "X(MAP1(s,pa_MAP)) == X(s);"
for each "X" in {"SSYNM" "awaitSSYNMrsp"}
  "X(MAP1(s,pa_MAP)) == true;"
Vm(MAP1(s,pa_MAP)) == Vm(s)+1;
Va(MAP1(s,pa_MAP)) ==
  if Vsc(s) then Va(s) else Vm(s);
for each "x" in {"mi" "ma" "dk"}
  "OWNED(MAP1(s,pa_MAP),x) == OWNED(s,x);"
for each "x" in {"FD" "HD" "SY" "MA"}
  "FU(MAP1(s,pa_MAP),x) == FU(s,x);"
SSP4(MAP1(s,pa_MAP)) ==
  put(put(put(null,name,_SSYNMind),
    sn,get(pa_MAP,sn)),
    data,get(pa_MAP,data));
for each "X" in {"SSP1" "SSP2" "SSP3" "TSP"
  "SPDU1" "SPDU2" "SPDU3" "SPDU4" "SPDU5"}
  "X(MAP1(s,pa_MAP)) == null;"
```

図3.2 コンパイラへの入力仕様(一部)


```

MAP(pa_MAP)
pdu5 *pa_MAP;
{
    if (phase == idle_and_TCcon)
    {
        err1();
    }
    else
    {
        if (phase == data_trans)
        {
            if (!SSYNM) && p12() &&
                p19(pa_MAP->sn))
            {
                MAP1(pa_MAP);
            }
            else
            {
                err0();
            }
        }
        else
        {
            if (phase == abort)
            {
                err0();
            }
        }
    }
}

```

(a)

```

MAP1(pa_MAP)
spdu5 *pa_MAP;
{
    SSYNM = TRUE;
    awaitSSYNMrsp = TRUE;
    if (Vsc)
    {
    }
    else
    {
        Va = Vm;
    }
    if (SSP4 == NULL)
        SSP4 = (ssp4 *)malloc(sizeof(ssp4));
    SSP4->name = _SSYNMind;
    SSP4->sn = pa_MAP->sn;
    SSP4->data = pa_MAP->data;
    SSP1 = NULL;
    SSP2 = NULL;
    SSP3 = NULL;
    TSP = NULL;
    SPDU1 = NULL;
    SPDU2 = NULL;
    SPDU3 = NULL;
    SPDU4 = NULL;
    SPDU5 = NULL;
    Vm = Vm + 1;
}

```

(b)

図3.3 生成されたCプログラム

3.5 結言

抽象的順序機械の形で記述された通信プロトコルの代数的仕様から手続き型プログラムへの変換法の形式的記述と、その方法に従って作成されたコンパイラについて述べた。今後、通信プロトコル以外の具体的な例題について本章で示した方法を適用すると共に、コンパイラの機能の拡張と改善を図る予定である。

結 論

本論文は、通信プロトコルの等価性及びエラーリカバリー性の検証とプログラムへの変換に関する研究をまとめたものである。

(1) 等価性の検証に関しては、2つのプロトコルマシン M_1 , M_2 間の Ψ 等価性を定義し、それらが Ψ 等価であるかどうかを判定する手続き与えた。

また、その手続きを実際の装置例に適用し、等価性（従って M_1 , M_2 の「互換性」）の証明において本手続きが有効であることを確かめた。

(2) エラーリカバリー性の検証に関しては、等価性の検証に用いたモデルを適用して、2つのプロトコルマシン M_1 , M_2 が検証者が与えた P 文 Φ に対して、エラーリカバリー性を持つ事を保証するための自動検証法を提案した。

また、その方法が HDLC 手順程度の実用規程の問題に適用可能であることを確かめた。

なお、この自動検証法は通信プロトコルの安全性や生存性の検証にも利用できる。本論文で述べた検証法がどの程度まで実用規程の問題に適用可能かどうかやその評価が今後の課題である。

(3) 通信プロトコルのプログラムへの変換に関しては、抽象的順序機械の形で記述されたプロトコルの代数的仕様から手続き型プログラムへの変換法とその方法に従って作成されたコンパイラについて述べた。

今後、プロトコル以外の具体的問題についてここで

示した方法を適用すると共に、コンパイラの機能の拡張と改善を図る予定である。

謝 辞

本研究に関して、直接理解ある御指導を賜り、さらに論文提出の機会を与えて頂いた大阪大学基礎工学部情報工学科の谷口健一教授に心から感謝申し上げます。

また、情報工学科の高忠雄教授には筆者が大阪大学基礎工学部大学院に在学中より御指導を賜り、本研究の全過程を通じてつねに励まして頂きました。ここに心より感謝の意を表します。

本研究に関して、終始、適切な御指導と御助言を頂いた東野輝夫講師並びに関浩之助手に心から感謝します。

本論文の作成に際し、有益な御助言を頂いた大阪大学情報工学科 都倉信樹教授、宮原秀夫教授、大阪大学教養部の藤井護教授、並びに滋賀大学情報管理学科の森将豪助教授に心から感謝します。

また、通信プロトコルの実現法とテスト系列に関して種々御教示頂いた慶応大学 松下温教授並びに沖電気工業㈱の新田哲二氏に心から感謝します。

さらに、等価性、エラーリカバリー性の検証に際し検証手続きに相当するプログラム作成を担当頂いた大学院生（現在 住友電気工業㈱ 勤務）の木本智久氏に感謝します。

- (1) 野口, 斉藤, 白鳥: "分散処理とコミュニケーション", ソフトウェア工学ハンドブック(榎本編) オーム社, pp177-221(1986).
- (2) J.E.Hopcroft and J.D.Ullman: "Introduction to Automata Theory, Language, and Computation", Addison-Wesley, (1979).
2-計数機械の定義, 及び, 2-計数機械が フーリング機械と等価であることについてはpp.171-173を, フーリング機械の等価性が判定不能であることについてはp.150及びp.281を参照. 訳書(野崎他訳) "オートマトン 言語理論 計算論I,II", サイエンス社(1984) では上記ページは各々巻Iのpp.224-226, 及び巻Iのp.196, 巻IIのp.82に相当.
- (3) 木本, 東野, 谷口, 二宮: "あるクラスの通信制御装置のモデル化と等価性判定について", 言語理論とオートマトンシンポジウム(1987-07).
- (4) 木本智久: "あるクラスの通信制御装置のモデル化と等価性判定", 大阪大学基礎工学部情報工学科特別研究報告(1987-03).
- (5) 竹内外史: "数学基礎論の世界", 日本評論社(1972).
- (6) 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", 信学論(D), J67-D, 4, pp.472-479(昭59-04).
- (7) "IBM Binary Synchronous Communications", IBM N:GA27-3004-1(1969).
- (8) "IBM 同期データ・リンク制御概説書", IBM N:GA27-3093-0(1975).
- (9) "SDLC(2次局)プロトコル仕様書", 沖電気工業(株) (1979).

- (10)野口, 斉藤, 白鳥: "分散処理とコミュニケーション", ソフトウェア工学ハンドブック(榎本編) オーム社, pp177-221(1986).
- (11)J.E.Hopcroft and J.D.Ullman: "Introduction to Automata Theory, Language, and Computation", Addison-Wesley, (1979).
- (12)木本, 東野, 谷口, 森, 二宮: "通信プロトコルにおけるエラーリカバリ性の検証の一方式", 信学技報IN88-80(昭63-09).
- (13)東野, 谷口: "整数線形計画問題の解非存在性判定を利用した通信プロトコルの自動検証について", 京都大学数理解析研究所講究録695, pp.243-252 (平01-06).
- (14)"JIS ハイレベルデータリンク制御手順", JIS C 6363-6365(昭53-11).
- (15)木本 智久: "通信プロトコルにおけるエラーリカバリ性の自動検証", 大阪大学大学院基礎工学研究科修士学位論文(平01-02).
- (16)加藤, 東野, 谷口, 森, 二宮: "通信プロトコルにおけるエラーリカバリ性自動検証の一方式", 第39情処全大, 4T-1, pp.1961-1962 (平01-10).
- (17)稲垣康善, 坂部俊樹: "抽象データタイプの代数的仕様記述法の基礎", 情報処理, 25 (昭59).
- (18)杉山裕二, 谷口健一, 嵩忠雄: "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).

- (19) T. Higashino, M. Mori, Y. Sugiyama, K. Taniguchi and T. Kasami: "An algebraic specification of HDLC procedures and its verification", IEEE Trans. Software Eng., SE-10, 6, pp.825-836 (1984).
- (20) 栗屋英司, 遠一光, 関浩之, 藤井護, 嵩忠雄: "OSIセッション層の代数的記述とその検証", 信学論(D-1), J72-D-I, 4, pp.272-283 (平1-04).
- (21) 田中哲雄, 谷口健一, 奥井順: "スクリーンエディタの代数的仕様記述とその実現", 信学論(D), J71-D, 7, pp.1207-1217(昭63-07).
- (22) K. Torii, Y. Morisawa, Y. Sugiyama and T. Kasami: "Functional programming and logical programming for the telegram analysis problem", Proc. of the 7th Intl. Conf. on Software Eng., pp.463-472 (1984).
- (23) 嵩忠雄, 谷口健一, 杉山裕二, 関浩之: "代数的言語ASL/* -意味定義を中心に-", 信学論(D), J69-D, 7, pp.1066-1074 (昭61-07).
- (24) 二木厚吉, 外山芳人: "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(昭58-02).
- (25) ISO: "Basic connection oriented session protocol specification", ISO 8327.
- (26) 井上克郎, 関浩之, 谷口健一, 嵩忠雄: "関数型言語ASL/Fとその最適化コンパイラ", 信学論(D), J67-D, 4, pp.458-465 (昭59-04).
- (27) A.V. Aho, R. Sethi and J.D. Ullman: "Compilers, Principles, Techniques, and Tools", Addison Wesley (1986).

- (28) 犬伏健二: "代数的に記述された通信プロトコルからプログラムへの変換システムの試作", 大阪大学基礎工学部特別研究報告 (1988-03).
- (29) 遠一光, 栗屋英司, 関浩之, 藤井護, 二宮清: "抽象的順序機械の形で記述された代数的仕様からプログラムへの変換について", 信学論(D-1), J73-D-I, 2, pp.201-213 (平2-02).
- (30) ISO: "LOTOS, a formal description technique based on the temporal ordering of observational behaviour", DP 8807.
- (31) ISO: "Estelle, a formal description technique based on an extended state transition model", DP 9074.
- (32) G.V. Bochmann, G.W. Gerber and J.M. Serre: "Semiautomatic implementation of communication protocols", IEEE Trans. Software Eng., SE-13, 9, pp.989-1000 (1987).
- (33) J.P. Briand, M.C. Fehri, L. Logrippo and A. Obaid: "Executing LOTOS specifications", Protocol Specification, Testing, and Verification, VI, pp.73-84, North-Holland (1987).
- (34) 加藤聰彦, 長谷川亨, 堀内浩規, 鈴木健二: "EstelleからAdaへのトランスレータ", 信学技報, IN86-97 (1986-11).
- (35) 高橋薫, 白鳥則郎, 野口正一: "LOTOS解釈機構の構成", 信学技報, IN87-107 (1988-01).

