順序機械型プログラムの階層的設計と分散実行プログラム群への変換

岡　野　浩　三

平成７年１月

順序機械型プログラムの階層的設計と分散実行プログラム群への変換

岡野 浩三

平成 7 年 1 月

内容梗概

本論文は, 筆者が大阪大学大学院 基礎工学研究科および同大学院基礎工学部に在学・在職中, 谷口 研究室において行ったソフトウェア工学の研究のうち (抽象的) 順序機械型モデルによるプログラムの階層的設計法, 分散動作仕様群の自動導出に関する研究をまとめたものである.

大域変数を表す内部レジスタの同時更新と入出力動作を一状態遷移で行なう順序機械型モデルを用いて, 多くの実用システムを記述することができる. 一般に高信頼システムを設計する際, 抽象レベルから具体レベルに順次詳細化していく方法が有効である. しかしながら, 抽象度の高いレベルにおける要求記述の記述法, 詳細化の正しさの証明法に関する研究はあまり行なわれていない. また, 分散システムを設計する場合, 抽象レベルではいくつのシステム (ノード) で分散システムを構成するかは考慮にいれず, 一つの制御部からなる順序機械として設計・詳細化し, 適当なレベルで分散システムに変換できることが望ましい. 最終的な実行形態が異なっていても, 抽象レベルでは, 単一の順序機械型モデルとしてとらえることができる. そこで, 本論文では, 順序機械型モデルに対して, 以下の二点について述べる.

第一に, 代数的言語 ASL を用い, 階層的設計の有用な枠組として, 拡張射影, 到達正当性条件を用いた記述法, 及び, これらを用いた記述の詳細化の定義, 詳細化の正しさの証明法を提案する. 最上位の要求を記述する場合, 以降の詳細化の自由度を大きくするためには, プログラムの入力系列と出力系列の満たす関係のみを指定すべきである. しかしながら, 入力系列と出力系列の満たす関係を直接基本述語を用いて表せない場合が多い. このとき, 補助的な内部レジスタを想定し, 入力系列と補助的な内部レジスタ, 及び, 内部レジスタと出力系列, それぞれの満たすべきの関係を記述すると自然に表せる. しかし, 補助的な内部レジスタは以降の詳細化の自由度を上げるために隠すべきである. 以上のことを自然に記述する枠組として拡張射影の概念を導入する. 拡張射影は, 興味ある関数の入出力値の対応を表す等式集合の集合を指定する. 拡張射影を用いた記述に対する詳細化の正しさの証明法として, 上位で補助的に用いた内部レジスタを表す関数を下位の (複数の) 内部レジスタを表す関数に対応させる関係を検証者が与えて, そのもとで, 項書き換え系, 整数上の加減算の論理式への帰着, 帰納法等を利用する方法を与える. したがって, 等式集合の集合を指定しているため,

単一の等式集合を指定する (拡張射影を用いない) 記法に比べ, 一般に証明作業の増大が予想されるにもかかわらず, 複雑な証明作業を要求するものではない. 一方, 到達正当性条件を用いた記述スタイルでは, 初期状態から正当な入力系列を与えたときに到達可能な (内部レジスタの状態も含めた) 抽象状態であれば真となる述語 (到達正当性条件) を定義し, この述語を前提条件として状態遷移の要求記述を行なう. この記述スタイルにより, 状態遷移の要求記述の際に具体的な前提条件を考慮しなければならない不都合や, 前提条件が緩すぎて実現に対する自由度を制限してしまう不都合, あるいは, 誤った条件を書いて記述全体が意味のないものとなる恐れがなくなる. 到達正当性条件を用いた記述に対する詳細化の正しさの証明法として, (記述の時点ではなく) 証明の時点で, 到達正当性条件が真なら成り立つと思われる具体条件を検証者が与え, 証明を行う方法をとる. この証明法における論法, 手間等は前提条件を具体的に記述するスタイルと比べほぼ同じである. また, 以上の二つの概念をプログラム設計の公開問題である在庫管理の階層的設計に適用し, これらの有効性を確かめた.

第二に, 信頼できる通信環境と信頼できない通信環境における分散システムの動作仕様の導出アルゴリズムを与えた. 順序機械型モデルによる単一のプログラムを, 複数のノードからなる分散計算システムの抽象的な全体仕様とする. 入出力動作を行うゲートと内部レジスタの各ノードへの割り当てを設計者が指定する. この全体仕様と割り当てをアルゴリズムの入力とし, 信頼できる通信環境において, 割り当てを満たし, 与えられた全体仕様とおりに動作する各ノードの動作仕様を出力とする. このアルゴリズムでは, 全体仕様の一状態遷移ごとこれを模倣する各ノードの状態遷移系列を導出する. この際, この導出法で採用している模倣方針のもとで, 全体仕様の一状態遷移あたりに必要なメッセージの送受信数を最小にする. 最小解を得るために 0-1 整数線形計画問題の解法を利用する. 信頼できない通信環境に対する導出アルゴリズムでは, 高々一箇所リンク故障を起こすような通信環境を仮定する. リンク故障によるメッセージ損失を回避し, かつ高速に動作する方法として同一情報を持つ複数のメッセージを異なる経路で送受信する方法をとる. 全体仕様の一状態遷移あたりに必要なメッセージの送受信数を最小にするために同様に 0-1 整数線形計画問題の解法を利用する. いずれのアルゴリズムも 10 数個程度のレジスタ数, 数個程度のノード数であれば, 実用的な時間内で動作仕様群が自動生成できることを確認した.

関連発表論文

(1). 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: “拡張有限状態機械モデルを用いた分散システムの要求記述から各ノードの動作仕様の自動導出”, 情報処理学会論文誌, Vol. 34, No.6, pp.1290-1301, (1993).

(2). 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “順序機械型プログラムの階層的設計法と在庫管理プログラムの開発例”, 電子情報通信学会論文誌 D-I, Vol. J76-DI, No.7, pp.354-363, (1993).

(3). 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: “リンクの故障を考慮に入れた分散システムの動作仕様の自動導出”, 情報処理学会論文誌, Vol. 36, No.1, pp.70-83, (1995).

(4). 岡野 浩三, 東野 輝夫, 谷口健一: “あるスタイルに基づく順序機械型記述における詳細化の正しさの証明方法”, 電子情報通信学会論文誌 D-I, (条件付き採録).

(5). Teruo HIGASHINO, Kozo OKANO, Hiroshi IMAJO and Kenicihi TANIGUCHI: “Deriving Protocol Specifications from Service Specifications in Extended FSM Models”, Proc. of 13th IEEE International Conference on Distributed Computing Systems, pp. 141-148, (1993).

(6). 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “代数的言語 ASL で記述した在庫管理プログラムとその正しさの証明”, 電子情報通信学会技術報告, SS 90-29, pp. 89-98, (1990).

(7). 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: “拡張有限状態機械モデルを用いた分散処理システムの全体記述から各ノードの動作記述の自動生成”, 情報処理学会研報告, 92–PRG–8, 27, pp. 211-218, (1992).

(8). 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: “リンクの故障を考慮に入れた分散システムの動作仕様の自動導出 ”, 電子情報通信学会情報基礎論ワークショップ LA シンポジウム報告, 93–LA, 68, pp.70-75, (1993).

目 次

1 緒論

大域変数を表す内部レジスタの同時更新と入出力動作を伴う順序機械モデル (以降は単に “順序機械型モデル” と呼ぶ) は, ソフトウェア, ハードウェア, 通信プロトコル等の抽象レベルの要求記述を行なう際の有効なモデルの一つである. 一般に高信頼システムを設計する際, 抽象レベルから具体レベルに順次詳細化していく方法が有効である. ある程度具体的なレベルまで詳細化した後では, 単体のソフトウェアとして実行する, あるいは同期式のハードウェアとして実装するためのシステム (コンパイラ, ハードウェア設計における論理合成系) が比較的研究されており, またいくつかの実用システムが知られている.

しかしながら, 抽象度の高いレベルにおける記述の枠組に関する研究, すなわち, どのように抽象度の高いレベルの記述を行なうべきか, どのように詳細化の正しさを証明すべきかに関する研究はあまりなされていない. また, 分散システムを設計する場合, 抽象レベルでは, いくつのシステム (ノード) で分散システムを構成するかは考慮にいれず, 一つの制御構造をもつ順序機械として設計・詳細化し, 適当なレベルで分散システムに変換できることが望ましい. この変換を自動的に行なうことができれば, ノード間でデータ交換や同期通信のために行なわれる繁雑なメッセージ交換を仕様記述者がいちいち記述する必要がなくなり, 設計の効率化, 高信頼化が期待できる.

本論文では, 順序機械型モデルに対して, 次の二点について述べる. (1) 代数的言語で, 順序機械型モデルを要求仕様からプログラムレベルまで段階的に詳細化する際の有用な記述法及び証明法を与える. (2) 順序機械型モデルで記述された単一の実行プログラムを, 複数のノードからなる分散計算システムの全体仕様とみなす. 内部レジスタ, 及び, 入出力動作を行なうためのゲートの各ノードへの配置指定と上述の全体仕様を入力とし, これらを満たし, かつ, 効率良く動作する各ノードの動作仕様を出力とするアルゴリズムを与える. (1),(2) の研究により, 順序機械型モデルにおける抽象レベルの階層的設計法がより有用となり (抽象レベルにおいては, 最終的な実行形態が単体のソフトウェア, ハードウェア, あるいは分散実行系であれ, 単一の順序機械型モデルとしてとらえることが可能である), また, 実行効率の良い分散システムを各ノードのプログラムレベルまで詳細化する設計方法が確立できる.

上述の (1) に関して, 代数的言語は主としてデータ抽象化の強力な手法として研究され

てきた[1, 3, 6, 5, 7]. これらのデータ抽象化では, より下位の基本的なデータ型から始めて, 複雑なデータ型を構成したり[2], いくつかの仕様構成子 (export, rename, hidden 等) の適用によって仕様を変換していく[1]というアプローチが多い. 一方, いくつかの研究では, 仕様の詳細化を形式的に定義し, 詳細化の正しさを形式的に保証するというアプローチを種々の形式モデルで行っている. たとえば, Z 記法[8]では, 形式的記述を行なうために集合等のいくつかの数学概念を用いる記述法を与えている. また, プロセス代数や, 時相論理を用いて, ハードウェアの実現の正しさを検証法まで含めて議論しているのもいくつか報告されている[10, 11]. しかしこれらは, 仕様記述を形式的に行うことを主たる目的としたものや, ハードウェアの下位レベルの記述とその実現のように比較的狭い範囲においての有効性を示したものといえる. また, 要求記述の際の性質記述に高階論理のような複雑なものを使っているアプローチ[9]もある. このように複雑な体系を用いると表現能力はあがるけれども, 証明支援系の各機能が複雑になり実用時間で処理できるかどうか問題である.

本研究では, 代数的言語としては一番単純な等式論理 (とほぼ等価な体系) に基づく意味論を持つ 代数的言語 ASL[17]を用い, このような単純な言語で実際にプログラムの階層的設計を行なう際に特に記述上有用である概念として (1-a) 拡張射影, (1-b) 到達正当性条件を用いた記述法を提案する. また, (1-c) これら概念を, プログラム設計の公開問題である在庫管理の階層的設計に適用し, これらの有効性を確かめた.

(1-a) 最上位の要求を記述する場合, 以降の詳細化の自由度を大きくするためには, プログラムの入力系列と出力系列の満たす関係のみを指定すべきである. 例えばソートプログラムの場合は, 「入力と出力は bag(要素の重複を考慮に入れた集合) として等しい. 出力は昇順になっている.」というように書くのが良い. この例では簡単にかつ自然にそれらの関係を記述することができるが, 在庫管理のように状態機械としてモデル化するほうが適しているような例題では入力系列と出力系列の満たす関係を (内部レジスタ値に関する) 状態の概念抜きに自然に記述するのは難しい. 在庫管理の例では, 出力である出庫指示書や出庫不可連絡の値は, 入力である出庫依頼書や積荷票と内部レジスタである在庫コンテナ情報や未出庫依頼書情報との相互関係で表すのが自然である. しかし一方では, 以降の詳細化の自由度を上げるためこれらの内部レジスタに関する具体的記述は実現に対する要求としたくない.

また, 入出力とこれらの内部レジスタ間の関係を記述する際にも以降の詳細化の自由度を上げるために, 値を一意に定めたくない. 例えば,「在庫コンテナ情報として保持しているコンテナのうちどのコンテナから出庫するか」等は一意に定めたくない. このようなことを満たすためには, 補助的な内部レジスタを導入し, かつ, 値を定めずに自由度を持たせて書いた記述を, 入出力に対する要求だけを自由度を持たせて書いた記述と見なす仕組みがあれば良い. このようなことができる枠組として, 本論文で拡張射影の概念を導入する. 拡張射影は, ASL における射影の概念の拡張である. 射影では表現式の合同関係を指定するテキストから, 興味ある関数に対して, 関数の引数と関数値の等式集合だけをとりだすことができる. この機構を用いると, 補助的な内部レジスタに関する具体的記述を実現に対する要求から外すことができる. しかしながら, 上述のように「入出力系列と補助的な内部レジスタの満たすべき関係」を記述したテキストでは, 興味ある関数値 (入力系列に対する出力系列の値) は一意に定めまらないので, 射影は適用できない. 拡張射影は, 補助的に用いる内部レジスタや関数を仮に定め, これらを用いて関数 Fの満たすべき条件を定めたテキストを参照して関数 Fの入出力値の対応を表す満たすべき条件に適合する全ての等式集合の**集合**を指定する. このことにより, より自由度の高い要求仕様を意味上の厳密さを欠くことなしに記述することが可能となる. 本論文では, 拡張射影を用いている記述 t_1, t_2に対して, t_2が t_1の正しい詳細化であることの定義を行ない, t_2が t_1の正しい詳細化であることを証明するための十分条件, および, 具体的な方法論を与える. この十分条件は, 以下の証明方法で証明し, それが成功したときに正しい詳細化が保証されるための条件である. 証明方法として, t_1で拡張射影を用いる際に補助的に用いた内部レジスタを表す関数を t_2で用いている複数の内部レジスタを表す関数で表すための対応関係を検証者が与えて, そのもとで従来の証明方法を適用する方法をとる. したがって, 拡張された記述クラスに対しても従来の証明方法[18] (項書き換え系の利用, 整数上の加減算を含む論理式への帰着, 帰納法) が適用でき, 証明作業量は増加しない.

(1-b) 到達正当性条件を用いた記述方法では, 順序機械型プログラムの階層的設計での各レベルの記述において, 各状態遷移前の内部レジスタが満たしているべき前提条件を陽に記述せず, そのレベルのプログラムの実行制御において初期状態から正当である入力系列が与えられたとして, 実際に到達可能な (内部レジスタの状態も含めた) 抽象状態であれば真と

なる述語 (到達正当性条件) を定義し，その述語を前提条件にして状態遷移の要求記述を行なうスタイルをとる．すなわち 到達正当性条件は, 実行の制御の条件と外部入力の正当性を表す述語の論理積だけで表されており, また, 具体的な内部状態に関する述語ではない．このスタイルをとると，記述の際，前提条件をいちいち考慮しなければならない不都合や，条件が緩すぎる場合に実現に対する自由度を制限してしまう不都合，あるいはそのレベルのプログラム実行を考えると成立しない誤った条件を書いて記述全体が意味のないものとなるといった恐れがなくなる．到達正当性条件による記述では，従来の合同関係の細分という概念では実用上妥当な詳細化を定義することはできない．そこで，言語自体の意味定義を複雑に変更する等といったことを避け, 従来の意味論で議論できるように, 上位の 到達正当性条件に関する定義 として「到達正当性条件と実行の制御の条件, 及び, 外部入力の正当性が成り立つときかつそのときのみ以降の状態でも正当である」ことを表す公理群の同値関係「かつそのときのみ」を含意関係「ならば」に読み変えたうえで従来の意味での詳細化関係が成り立てばよい, と定義する．また，正しい詳細化であることの証明方法を与える．本証明法では, (記述の時点ではなく) 証明の段階で, 到達正当性条件が真なら成り立つと思われる内部レジスタに関する具体的な条件を検証者が与え，それをもとに (不変式であるという証明も含め) 正しい詳細化であることの証明を行う方法をとる．この証明における論法，手間等は従来の前提条件を具体的に記述するスタイルと比べほぼ同じである．

(1-c) プログラム設計の例題として提供された在庫管理問題[12, 13, 14]に適用して, これらの概念の有用性を調べた．その結果, 在庫管理問題を入出力の関係のみを指定した要求仕様から具体的なプログラムレベルまでこれらの概念を用いて階層的に記述できることが示された．我々のアプローチとほぼ類似のアプローチをしているイオタ[15]との比較も簡単に行なった．

上述の (2) に関しては, 主としてプロトコル設計の分野で精力的に研究がされてきた．一般に複数のノードが同期や, データ交換のためにメッセージを交換しながら, 全体として意味のある動作をするシステムを設計する際, 前述のように, メッセージ交換を含む各ノードの動作仕様を記述するのは繁雑であり, また間違いも生じやすい．そこで, (ノード間のメッセージ交換を無視し) システム全体としての外部から観測できる動作とその実行順を表した全体

仕様を設計者が記述し, そこから各ノードの動作仕様を導出する方法が研究されてきた[21]. これらの研究は, 有限状態機械モデルに基づく研究[27, 28]と, 形式的記述言語 LOTOS[26]を用いた研究[22, 23, 24, 25]がある. LOTOS モデルでは, 非決定性の選択動作や, 並列実行を記述できるが, 内部データを保持するレジスタを分散配置できない欠点があった. 他方, 有限状態機械モデルではメッセージの消失への対応を考慮に入れた研究がいくつかあるが, データを陽に扱う研究はほとんどなされていない. 有限状態機械モデルでデータを取り扱えるようにモデル拡張をする場合, データを保持するレジスタの配置を適宜に指定することにより, 分散実行に適する実行システムを作ることができる. 本研究であつかう順序機械型モデルは有限状態機械モデルにデータを扱えるよう拡張したものである. このモデルに対して, 信頼できる, あるいは, 信頼できない通信環境における各ノードの動作仕様を自動導出するアルゴリズムを以下のように考案した.

(2-a) 順序機械型モデルのプログラムを, 複数のノードからなる分散計算システムの抽象的な全体仕様とみなす. 入出力動作はゲートを介して行なわれるものとし, 入出力動作を行うゲートと内部レジスタの各ノードへの割り当てを設計者が指定するものとする. この全体仕様と割り当てをアルゴリズムの入力とし, 信頼できる通信環境において, 指定された割り当てを満たし, 全体仕様どおりに動作する各ノードの動作仕様を出力とする. このアルゴリズムでは, 複数のノードに配置されたレジスタ値の一貫性を保つために以下のように, 全体仕様の一状態遷移ごとこれを模倣するように各ノードの状態遷移系列を導出する. 各ノードの状態遷移系列は, 状態遷移の決定と入出力動作, 選んだ状態遷移名の通知, (レジスタ更新の引数となる) レジスタ値の転送, レジスタ値の更新, (次状態遷移の条件判定や出力動作に用いられる) 新レジスタ値と同期メッセージの通知の 5 つのフェーズからなる. レジスタ値の更新の際, 更新式の引数である各レジスタ値を (必要であれば) 上記のレジスタ値の転送のフェーズでメッセージとして受け取り, その後各ノードで更新を行なう. また, レジスタ更新を行なったノードはすべて, 同期メッセージの通知を行なう. 次の状態遷移の模倣は, すべての同期メッセージが受信されるまで開始しない. これによりレジスタ値の一貫性が保証できる. 交換されるメッセージに入力値のデータが付加することもありうる. また, 状態遷移の通知や同期メッセージの通知の際に交換されるメッセージに, レジスタ値等の情報が付加さ

れることもある. 本アルゴリズムでは, 一状態遷移あたりに必要なメッセージの送受信数をこの模倣方針のもとで最小にする. メッセージ長はコスト基準にはいれない. このとき, どのフェーズでどの内容 (レジスタ値や入力値) をどのノード間で送受信するかについては自由度が生じる. これらの自由度を考慮して最小解を得るために 0-1 整数線形計画問題の解法を利用する.

(2-b) 高信頼性が要求される分散システムにおいては, 通信ネットワークに故障が生じても, 柔軟に対応できる必要がある. ここではリンク故障を起こすような通信環境として, (i) リンク故障によりメッセージが消失する, (ii) リンクの故障は高々一箇所だけである, 等を仮定する. リンク故障数に上限があれば, 同一情報を持った複数のメッセージを異なる経路で送受信する方法がリンク故障の回避方法として考えられる. 一般には故障回避方法としてタイムアウト機構を用いる方法がとられるが, タイムアウト時間が大きく取られるため実行効率はあまりよくない. この点で, リンク故障が頻繁に起こらないような通信路では, タイムアウト機構を用いるよりも同一情報を持った複数のメッセージを異なる経路で送受信する方が高速に動作できる. 本論文では, 後者の方法を採用し, この方法で動作する動作仕様群を自動導出するアルゴリズムを考案した. レジスタ値の更新の一貫性の保証や, 各状態遷移の同期の保証は (2-a) と同様に, 一状態遷移ごとこれを模倣することで行なう. この際, メッセージの重複をできるだけ小さくするように, 0-1 整数線形計画問題の解法を利用する.

いずれのアルゴリズムも 10 数個程度のレジスタ数, 数個のノード数であれば, 0-1 整数線形計画問題を総当り的にとく解法を用いても, 実用的な時間内で解けることを確認し, 本方式による分散実行動作仕様群の導出アルゴリズムの有用性がわかった.

以降, 2 章で 順序機械型モデルおよび, 代数的言語 ASL について簡単に述べる. 3 章で順序機械型モデルにおける階層的設計法と拡張射影, 到達正当性条件の概念と詳細化の正しさの証明法について述べる. また, 在庫管理問題を例題として, 本手法を適用した結果について簡単に述べる. 4 章で 分散動作仕様群導出問題について諸定義を行ったのち, 動作仕様群の自動導出アルゴリズムについて述べる. 5 章ではリンク故障を持つ通信環境に対応する方法について述べる. 6 章でまとめと, 今後の課題について述べる.

2 諸準備

本章では, 順序機械型モデルについて簡単に述べた後, 代数的言語 ASL についてその基本概念を簡単に述べる.

2.1 順序機械型モデル

レジスタを持つ順序機械型モデルは以下の 4 字組 $M = (S, R, \delta, init)$ で定義する.

S, Rはそれぞれ有限制御部の状態の有限集合 $\{s_1, \ldots, s_n\}$, 有限個のレジスタの集合 $\{r_1, \ldots, r_m\}$ である. δは状態遷移関数の集合で, 各状態遷移関数は $s - CR \rightarrow s'$で表す. ここで CRはレジスタ更新のための代入文の集合で, その代入文をレジスタ更新式と呼ぶ. 各代入文の右辺の関数の引数には, レジスタ値が許される. $init$ は初期状態と各レジスタの初期値を指定する.

このモデルは以下のように動作する. ある状態において, その状態から出ている全ての遷移のうち 1 つを非決定的に選択する (遷移の選択). その後, レジスタの更新を行い, 次の状態へ移る. レジスタの更新は, 全ての代入文の右辺の値の評価を行った後, 全ての代入を同時に実行 (同時代入) する.

2.2 代数的言語 ASL

ここでは代数的言語 ASL[17]について簡単にのべる.

テキスト $t = (G, AX)$ は, 表現式集合を指定するための開始記号を持たない文脈自由文法 $G = (V_N, V_T, P)$ と, 公理集合 AXの対からなる. 文脈自由文法 G において, 非終端記号集合 V_Nの任意の要素から生成規則の集合 Pにしたがって生成される終端記号集合 V_T上の記号列のすべての集合を, 文法 G が指定する**表現式集合** $E(G)$ と呼ぶ. $p \in V_N$は型を表す.

V_N中の非終端記号 p に対して, V_Nと V_Tのいずれにも属さない記号 x_p を複数設け, 生成規則 $p \rightarrow x_p$を考える. この生成規則を拡大生成規則 P'と呼び, 拡大生成規則を G に追加して得られる文法を G'とする. G'の指定する表現式集合 $E(G')$ を**項集合**と呼び, その要素を**項**と呼ぶことにする. 記号 x_pは型 p の変数と考えられる. 項ξの各変数に対して**許される代入**ρは変数 x_{p_i} にはその型 (非終端記号)p_iから生成される表現式のみに限られるとする.

テキスト t の公理集合 AXは，$\{\xi == \eta \mid \xi, \eta \in E(G')\}$ なる集合である．各$\xi == \eta$を**公理**と呼ぶ．

テキスト t は，表現式集合 $E(G)$ とその上の合同関係$\equiv_t$を指定する．

[定義 1] [合同関係$\equiv_t$]

一つの公理$\xi == \eta$はξ, ηに対する任意の許される代入ρ によって得られる表現式$\rho(\xi)$ と $\rho(\eta)$ の等式集合を生成する．テキスト t の表現式集合 $E(G)$ 上の合同関係$\equiv_t$は，公理集合 AX中の各公理が生成する等式集合を含む最小の合同関係とする[17]． □

この合同関係は常に存在し一意に定まる[17]．

テキストのV_Nの部分集合V_{NC}を指定し，それに属する非終端記号 (型)Mから生成される表現式を型 Mの**構成子表現式**という．テキスト $t = (G, AX)$ 及び，項ξにおいて，項ξに対して許される任意の代入ρで定まる表現式$\rho(\xi)$ が，ある構成子表現式と合同関係にあるとき，項ξはテキスト t において値をもつ (値はその構成子表現式) と呼ぶことにする．以下，表現式集合と言うときなどは，構成子表現式が暗に指定されているものとする．

[定義 2] [合同関係をみたす関係]

E_1, E_2を $E_1 \subseteq E_2$なる表現式集合とし，$\equiv_1, \equiv_2$をそれぞれその上の合同関係とする．次の条件が成立するとき，合同関係$\equiv_2$は，合同関係$\equiv_1$をみたすといい，$\equiv_1 \subseteq \equiv_2$ (あるいは $\equiv_2 \supseteq \equiv_1$) と表現する．

E_1中の任意の表現式ξ, ηに対して，$\xi \equiv_1 \eta$ならば，$\xi \equiv_2 \eta$が成り立つ． □

[定義 3] [無矛盾]

表現式集合とその上の合同関係$\equiv$に対し，表現式集合中の相異なる二つの構成子表現式間に合同関係$\equiv$が成り立たないとき，合同関係$\equiv$は**無矛盾**であると言う． □

プログラムの仕様を書くためには，どの項の計算を実現するかという (興味ある項の) 概念が必要になる．

[定義 4] [正しい実現]

2つのテキスト $t_1 = (G_1, AX_1), t_2 = (G_2, AX_2)$ において，$E(G_1) \subseteq E(G_2)$ とする．また，テキスト t_1上の幾つかの項集合をτとする．次の条件が成立するとき，項集合τに関して，テキスト t_2はテキスト t_1の正しい実現であるという．

1. $\equiv_{t_1} \subseteq \equiv_{t_2}$が成り立つ．

2. τ中の各項ξとそれに対してt_1上で許される任意の代入ρに対して $\rho(\xi)$ は t_2において値をもつ．

3. $\equiv_{t_2}$は無矛盾である． □

直観的には，ある項$\xi \in \tau$に関して正しい実現をすれば，その正しい実現をしたテキスト上で合同関係により項ξの値を求めることができる．

射影[17]を用いることによって，引用したテキスト s の合同関係から興味ある等式集合のみ取り出せる．このような枠組みは，他の手法では hidden 関数等の概念が用いられている[1]．hidden 関数を用いたテキストの意味定義は，かなり複雑になる．我々の言語では，合同関係で同等の概念を定義する．これにより，簡明な意味定義が可能となる．

[定義 5] [射影]

テキスト t 中の射影文ξ == η@**text** s は，テキスト t 上でξ, ηに対して許される代入ρのうち，$\rho(\xi) \equiv_s \rho(\eta)$ を満たす表現式の対 $\langle \rho(\xi), \rho(\eta) \rangle$ の集合を等式集合として生成する．但し，s 中に 射影文はないとする．テキスト t の定める合同関係$\equiv_t$は各公理，射影文が生成する等式集合の族を含む最小の合同関係である[1]． □

[例 1]

例えば整数を逐次入力し，過去の入力系列中に現在の入力と同じ入力があれば，そのうちで最も先頭のものを，なければ 0 を返すプログラム P を設計する．このとき，**表 1**のテキスト impT のように，まず過去の入力系列を記憶する History 等の補助関数を用いて現在の入力 i に対し，出力 Out を得る状態遷移 T を定義する．この処理 T の入出力関係のみを射

[1]射影文による参照関係がサイクルにならない場合にも，もちろん合同関係は定義できる．ここでは簡単のため本定義を採用する．

影により取り出し，補助関数を仕様から隠すことができる (**表 2**テキスト specP の射影文). 初期状態から始めて状態遷移 T を入力系列に対して繰り返すというプログラム P を specP は指定している. □

2.3 ASL による順序機械型モデルの表現

ここで順序機械型記述を代数的言語 ASL で行なうために ASL の記述の部分クラスである ASM 記述[16]について述べる. 前述の**例 1**は ASM 記述の例でもある.

ASM 記述では, 抽象状態を表す State という型を導入し, この型の変数を引数にとる 状態成分関数と状態遷移関数を導入する.

順序機械型モデルにおける, レジスタと状態遷移が, それぞれ, 状態成分関数と状態遷移関数に相当する.

いま, あるテキスト t において, ASM 記述を行なうとき, 以下の指定を各項目ごと公理で行なう.

[初期化関数]

状態成分関数 $f_1, \ldots, f_n$に対して,

$f_i(\mathrm{init}(\vec{m})) == g_i(\vec{m});\ \ (1 \leq i \leq n)$

ここで, g_iは適当な基本関数である. これらの公理は適当な外部からの (複数の) 入力$\vec{m}$に基づいて, 各状態成分関数 f_iの初期値を定めることを意味している.

[状態遷移関数]

状態成分関数 $f_1, \ldots, f_n$に対して, 状態遷移関数 t は以下の公理形で指定する.

$f_i(t(s)) == h_i(f_1(s), \ldots, f_n(s));\ \ (1 \leq i \leq n)$

ここで, h_iは適当な基本関数である. これらの公理は状態遷移 t 後の各状態遷移関数の値を遷移前の各状態遷移関数の値から定めることを表す. 関数 h_iは **2.1節**の CRの式に相当する.

ASM 記述では, 状態及び, (状態間をつなぐものとしての) 状態遷移の指定を直接行なわず, 以下のループ関数と計算指定で行なう.

[ループ関数]

ループ関数 L を用いて, 複数の状態遷移関数 t_j間の実行順指定を以下のようにして行なう.

表 1: 状態遷移 T の仕様記述例 1

```
text impT;(* 処理 T,Init の実現 *)
(*初期化 (イ) の記述 *)
   Out(Init)==0;
   History(Init)==NILL;
   point(Init)==NILL;
(*処理 T のループ関数による展開 (ハ)*)
   T(s,n) == UL(UI(s,n),n);
   UL(s,n) == if point(s) = | History(s) | +1 then UE(s,n)
        else if get(point(s), History(s))= n
        then UF(s,n) else UL(UN(s,n),n);
(*処理 UI,UE,UF,UN の定義 (ロ)*)
   Out(UI(s,n))==Out(s);
   History(UI(s,n))==History(s);
   point(UI(s,n))==1;
   Out(UE(s,n))==0;
   History(UE(s,n))==add(History(s),n);
   point(UE(s,n))==point(s);
   Out(UF(s,n))==point(s);
   History(UF(s,n))==add(History(s),n);
   point(UF(s,n))==point(s);
   Out(UN(s,n))==Out(s);
   History(UN(s,n))==History(s);
   point(UN(s,n))==point(s)+1;
end impT;
```

表 2: 射影によるプログラム P の仕様記述例

```
text specP;
   intC I,O; stateC S;
   Out(T(S,I))==O @textimpT; (*処理 T の仕様*)
   Out(Init)==0; (*初期化 (イ)*)
include text main; (*main の内容のとりこみ*)
end specP;
text main;
project primitives; (* 基底代数 *)
   Out(L(Init,Int_Seq)); (*計算指定項 (ニ)*)
(*ループ関数 (ハ)*)
   L(s,int_Seq)== if int_Seq = NILL then s
         else L(T(s,car(int_Seq)),cdr(int_Seq));
end main;
```

$L(s) == K_1(s)$;

$L(s) ==$ if $p(s)$ then $K_2(s)$ else s;

ここで, K_iは状態遷移関数 t_jか, 状態遷移関数にループ関数を適用したもの $(L(t(s)))$ とする. また,$p(s)$ は, 状態 s における複数の状態成分関数値 $f_i(s)$ を引数にとる述語とする.

[計算指定項]

初期化関数, ループ関数, 及び, 状態成分関数を指定することにより, 初期状態から始めて実行終了後の (指定した) 状態成分関数値を求めるプログラムを指定することができる. この指定を計算指定項と呼ぶ.

$F(\vec{m}) == f_j(L(\text{init}(\vec{m})))$;

$F(\vec{m})$ は, 引数$\vec{m}$に対して, 初期状態 $\text{init}(\vec{m})$ から始めて, L の実行後の状態成分関数 f_jの値を返す関数 (計算指定項) である.

以下では, 上記, 初期化関数による状態成分関数の値の指定公理を (**初期化**の記述)(イ) とよぶ, 以下同様に, 各状態遷移関数の指定, ループ関数による実行順序指定の記述, 計算指定項の記述をそれぞれ, (**状態遷移内容の定義**の記述)(ロ), (**実行順序指定**の記述)(ハ), (**計算**

指定の記述)(ニ), と呼ぶ.

3 順序機械型モデルの階層的記述における有用な概念

3.1 序言

本章では, まず順序機械型の階層的設計法について簡単に述べた後, 拡張射影の概念についてのべる. 拡張射影は, ASL における射影の概念の拡張である. 射影の概念は一般の代数的仕様記述における 隠蔽 関数, 隠蔽 ソートの概念 (hidden function, hidden sort)[1, 7] に対応している. この章では, 拡張射影を用いている記述 t_1, t_2に対して, t_2が t_1の正しい詳細化であることの定義を行ない, t_2が t_1の正しい詳細化であることを証明するための十分条件および, 具体的な方法論を与える.

また, 到達正当性条件による記述方法を与え, この記述を行なったテキスト間の詳細化の定義を与える. 詳細化の正しさを保証する証明方法を与え, 従来手法との比較を述べる.

最後に, 拡張射影と到達正当性条件を用いて在庫管理問題を階層的に設計した記述例について述べる.

3.2 順序機械型記述の階層的設計

ASM の階層的設計スタイルを以下に簡単に述べる. あるレベルで導入された状態遷移関数は, 普通 (ロ) の変わりに **状態遷移関数の要求性質**の記述 (ホ) がなされる. (ホ) では, その状態遷移関数を適用する前後の各状態成分関数値がどのような関係にあるべきかが述語の形式で記述される. ある状態遷移の各状態成分関数の記述に述語形式と定義形式が混在してもよい. このとき, この状態遷移関数の記述は (ホ) の記述とみなす. 次のレベルでは, (複数の) より具体的な状態遷移関数を導入し, (ホ) の対象となった状態遷移の一部または全部は, 導入された状態遷移関数とループ関数を用いて, 詳細化される. 新たに導入された状態遷移関数については,(ホ) の記述がされる. 性質記述を書くまでもない簡単な状態遷移については, (ホ) の代わりに状態遷移内容の定義 (ロ) を記述する. 残りの記述は上位の記述をすべて受け継ぐものとする. よって上位で用いられた状態成分関数は原則として, 下位の記述でも用いられる. 但し, 拡張射影で用いられる補助的な状態成分関数は必ずしも下位で用いる必要はない. これは後述する. (ホ) がなくなるまで順次, 詳細化を続ける. 最下位の記

述から (イ), (ロ), (ハ),(ニ) を集めると実現プログラムとなる.

3.3 拡張射影

従来のテキストは合同関係を一つ定めていた. このように一つの合同関係で意味定義する方法は他の言語でも普通に行われている. 今後, 拡張射影を含むテキストを考えるが, これは合同関係の集合を表す. その集合のうちの任意の一つの合同集合を (従来の意味で) 満たすように実現すれば良いという意図である.

3.3.1 拡張射影の定義

以下では, テキスト x の表す表現式集合を E_xと表すことにする.

[定義 6] [拡張射影文を含むテキスト u の定める合同関係集合]

テキスト u 中の$\xi_1 == v_1$**@@text** s_1; $\xi_2 == v_2$**@@text** s_2; $\cdots$; $\xi_m == v_m$**@@text** s_n; なる文を拡張射影文という. ここで, 各 v_jは変数で, ξ_j中の変数や, v_jには構成子表現式だけを代入可能とする. また, テキスト s_k $(1 \leq k \leq n)$ 自体は, 拡張射影文を含まないものとし, $E_u \subseteq E_{s_k}$が成り立つとする. 上記の拡張射影文において同じ s_kを参照している項ξ_jは一般に複数存在する. s_kを参照している項の集合をτ_kとする. テキスト t_kをテキスト s_kの項集合τ_kに関する正しい実現であるとする. (t_kは一つの合同関係$\equiv_{t_k}$を定める. テキスト s_kと, τ_kを与えたときに, そのような テキスト t_kは一般に複数存在する.) 各 s_kとτ_kについて, そのようなテキスト t_kを一つ定めたとき, 拡張射影の 各$\xi_j == v_j$**@@text** s_k $(\xi_j \in \tau_k)$ を$\xi_j == v_j$**@text** t_kと読み変えて普通の射影とみなせば, テキスト u の合同関係が一つ定まる. この合同関係を $\equiv_{u[t_1,t_2,\ldots,t_n]}$と表す.

テキスト u の表す合同関係集合 $\mathcal{R}_u$は以下の通り.

$\mathcal{R}_u = \left\{ \equiv_{u[t_1,t_2,\ldots,t_n]} \middle|\ \text{各 } t_k \text{は } s_k \text{の項集合} \tau_k \text{に関する正しい実現} \right\}$

テキスト u が拡張射影を含まなければ $\mathcal{R}_u = \{\equiv_u\}$ とする. □

3.3.2 拡張射影を用いた仕様例

前章のプログラム P'の仕様は拡張射影を用いて, 以下のように記述できる.

表 3: 状態遷移 T の仕様記述例 2

text specST;
int n; state s;
AX1:Out(T(s,n))≠0 ⊃ ⟨n,Out(T(s,n))⟩ ∈ Set(s) == TRUE;
AX2:Out(T(s,n)) = 0 ⊃ ⟨n,c⟩ ∉ Set(s) == TRUE;
AX3:Set(T(s,n)) = Set(s) ∪{⟨n, |Set(s)| +1⟩} == TRUE;
AX4:Set(Init) == ∅;
include text main;
end specST;

[例 2]

specST に状態遷移関数 T の性質を入力変数と番号の対を要素とする集合を表す補助状態成分関数 Set を用いて 例えば,「状態遷移 T 後の Out の値が 0 と等しくなければ, n と Out の値の対が Set の要素として存在する.」等と書いておく (**表 3**). プログラム P' の仕様 specP' は, specP の射影文を拡張射影
Out(T(S,N))==O @@ **text** specST に変更したテキストである. □

3.3.3 拡張射影を含むテキストにおける詳細化の正しさの定義

[定義 7] [正しい詳細化]

2 つのテキスト $t_1 = (G_1, AX_1)$, $t_2 = (G_2, AX_2)$ において, $E(G_1) \subseteq E(G_2)$ とする. テキスト t_1の表す合同関係集合を $\mathcal{R}_1$とし, テキスト t_2の表す合同関係集合を $\mathcal{R}_2$とする. 次の条件を満たすときテキスト t_2はテキスト t_1の正しい詳細化であるといい, $t_1 \ll t_2$と表す.

合同関係集合 $\mathcal{R}_2$中のどの合同関係$\equiv_2$についても, 合同関係集合 $\mathcal{R}_1$中にある合同関係$\equiv_1$が存在して, $\equiv_2 \supseteq \equiv_1$ である. □

3.3.4 拡張射影を含むテキストにおける詳細化の正しさを保証する十分条件

ASM 型記述スタイルで記述したあるテキスト t とそれを ASM 型記述スタイルで詳細化したテキスト s について, 共に拡張射影を用いているとき, $t \ll s$ であるための十分条件

(**命題 1**) をのべる．まず，前提条件を置く．

[条件 1]

テキスト t,s はそれぞれ，テキスト w,zのみを拡張射影文で参照しており，各テキスト t,s,w,zは，基底代数に関する共通の射影文以外の射影を含まない．この基底代数に関する射影文の記述を Primitives とする．これは基本関数の (無矛盾な) 定義表を表す．

テキスト t (または s) における各状態遷移関数に関する記述は，議論の簡単のため以下の 3 つの記述のタイプのいずれかであるとする[2]

1. t (または s) に現れるすべての状態成分関数について，「定義」されている．
2. t (または s) に現れるすべての状態成分関数について，Primitives 中の基本関数を用いて「性質記述」されている．
3. t (または s) に現れるすべての状態成分関数について，以下の拡張射影文が記述されている．

その状態遷移を Tとする．すべての状態成分関数 Fについて，

$F(T(s,n)) == v$@@ **text** w (または s 上においては z)

テキスト t から，Primitives とタイプ 3 の記述を除いた残りを ASM Common とする．テキスト s は，テキスト t の タイプ 3 の記述の対象となった状態遷移関数のみを展開しているとする．また Primitives と ASM Common を含むものとする．

テキスト t,s の拡張射影文が参照しているテキスト w,zは以下の条件を満たす．テキスト w (または z) は，テキスト t (または s) のタイプ 3 の記述以外のすべての記述及び，次の記述からなる．テキスト t (または s) のタイプ 3 の記述の対象となった状態遷移関数に関するタイプ 1, 2 の記述．必要ならば補助的に用いる状態成分関数に関する記述を含んで良い．

テキスト iは以下の公理からなる．wで用いられた各補助状態成分関数に対して，s の (複数の) 状態成分関数でどのように対応づけるかを表す公理．これらは補助状態成分関数を s

[2]一般には，s,t において，一つの状態遷移の記述に関して，「定義」と「要求性質」のスタイルが混在する場合についても，正しい詳細化であるための条件を求めることは可能であるが，これらの一般化は本論文では省略する．

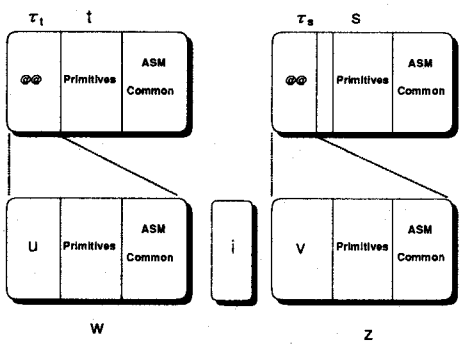


図 1: テキストの関係

Fig. 1 Text t and s

上でどのように実現するかを表す (ASL) テキストである．対応の記述に用いる関数はすべて，Primitives で定義されているものとする．

Primitives と ASM Common を合わせてテキスト common とする (以上 図 1)． □

テキスト t(または s) のタイプ 3 の記述の対象となった項の集合をτ_t(または, τ_s) とする．なお, これらの条件のもとでは, 表現式集合について, $E_t \subseteq E_s$が成立つ．テキスト i は，必ずしも (そのとおり) 実現しなくても良い補助状態成分関数が，下位のテキスト s でどのように実現されているかを表す公理群で，これは「正しい詳細化であること」の証明時に検証者が与える．

[定義 8] [定理]

テキスト t 上の項の対 $\langle \xi, \eta \rangle$ について t 上の許される任意の代入ρによって得られる表現式$\rho(\xi), \rho(\eta)$ に対し，$\rho(\xi) \in E_s$, $\rho(\eta) \in E_s$が成り立ち，かつ$\rho(\xi) \equiv_s \rho(\eta)$ であるとき，t 上の $\langle \xi, \eta \rangle$ は s 上で**定理**として成り立つと言い，$\xi \approx_s^t \eta$と表現する． □

以下の命題の証明のために「定理」について以下の補題[18] を用いる．

[補題 1]

テキスト t と s はともに拡張射影を含まず, 同じ基底代数に対する射影のみをともに含んでいるものとする．また, $E_t \subseteq E_s$を満たすものとする．

テキスト t 上のすべての公理$\xi == \eta$ について, $\xi \approx_s^t \eta$であれば, $\equiv_t \subseteq \equiv_s$ である. □

[命題 1]

条件 1の下で, テキスト $(z+i)$ 上でテキスト u の任意の公理が定理として成り立ち, かつ「zのτ_sに関する任意の正しい実現上で, τ_tの各項に t 上で許される任意の代入を行なって得られた項が値をもつ」ならば, $t \ll s$ である. □

証明: テキスト t, s の表す合同関係の集合をそれぞれ, $\mathcal{R}_t$, $\mathcal{R}_s$とする. 任意の$\equiv' \in \mathcal{R}_s$に対して, $\equiv' \in \mathcal{R}_t$を証明すれば十分である. まず, **条件 1**よりテキスト zと wはともに拡張射影及び射影を含まず, また同じ基底代数に対する射影のみをともに含んでいる. また, w, z, i の構成より, $E_w \subseteq E_{(z+i)}$が成り立つ. また, テキスト wの任意の公理は, u にあるか,common にあるかいずれかである. u にあるときは, **命題 1**中の条件「テキスト $(z+i)$ 上でテキスト u の公理が定理として成り立つこと」を用い, common にあるときは common の公理がテキスト $z+i$ にもあることを用いて, $E_w \subseteq E_{(z+i)}$と**補題 1**より

「$\equiv_w \subseteq \equiv_{(z+i)}$」$\cdots$ (*)

が導ける . また, **定義 6**より, 各$\equiv'$は,「E_s中の任意の表現式ξ, ηについて,$\xi \equiv_z \eta$ ならば $\xi \equiv' \eta$」を満たす.

さらに, iは, u 中の補助状態成分関数を独立に s 中の (したがって, z中の) 状態成分関数に対応させるだけなので, E_s中の項については, $\equiv_z$と $\equiv_{z+i}$とでは合同関係は変わらない. したがって,

各$\equiv'$は,「E_s中の任意の表現式ξ, ηについて,$\xi \equiv_{(z+i)} \eta$ ならば $\xi \equiv' \eta$」を満たす. $\cdots$ (**).

$E_t \subseteq E_s$, (*),(**) と, 合同関係の満たす関係の推移律より, 各$\equiv'$は,「E_t中の任意の表現式ξ, ηについて$\xi \equiv_w \eta$ならば$\xi \equiv' \eta$を満たす.」一方, **命題 1**の「zのτ_sに関する任意の正しい実現上で, τ_tの各項に t 上で許される任意の代入を行なって得られた項が値を持つ」とτ_tの各項に t 上で許される代入をして得られた任意の表現式が E_sに属することより, τ_t中の項が$\equiv'$上で値を持つ. ゆえに, $\equiv' \in \mathcal{R}_t$である. □

次に, 詳細化の正しさの証明法を上述の十分条件に基づいて与える.

3.3.5 拡張射影を含むテキストにおける詳細化の正しさの証明法

$E_t \subseteq E_s$であることは, 順序機械型の詳細化をしていれば成り立つので, $\equiv_t \subseteq \equiv_s$ であることを示すには, t 上のすべての公理$\xi == \eta$について, $\xi \approx_s^t \eta$であることを示せばよい.

命題 1よりただちに次の方法が得られる.

[方法 1]

条件 1の下で, (テキスト i を考案し) $t \ll s$ であることを保証するために, u の各公理 $\xi == \eta$について, $\xi \approx_{(z+i)}^u \eta$であることを確かめる.「$z$の$\tau_s$に関する任意の正しい実現上で, τ_tの各項に t 上で許される任意の代入を行なって得られた項が値を持つ」ことは, 最終的な実現プログラムを得てこの上で無矛盾性, 停止性を確かめることによって確認する. □

3.3.6 証明例

例 2に対する証明例をここで述べる. この例では下位の記述は拡張射影を含まないのでより簡単になる. 上位で集合 Set を用いて記述し, 下位ではリスト History を用いて実現しているので, これらの関係を与えるテキスト i の公理として例えば,

Set (s) == LS(History(s)) ,

のように, リスト History から 集合 Set への変換関数 LS を用いて表す. LS に関する基本補題として,

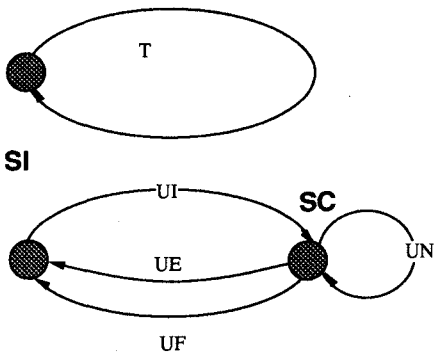
LS(add(l, n))= LS(l) $\cup\{\langle n, \mid l \mid +1\rangle\}$ == TRUE

LS(NILL) = $\emptyset$ == TRUE

$\langle n, i\rangle \in$ LS(l) $\Leftrightarrow n =$ get(i,LS(l))== TRUE

等があげられる.

テキスト (impT+main+i) の合同関係$\equiv_{(\mathrm{impT+main+}i)}$と, SpecST の合同関係$\equiv_{\mathrm{SpecST}}$について, $\equiv_{(\mathrm{impT+main+}i)} \supseteq \equiv_{\mathrm{SpecST}}$ であることが証明できる. また, OUT(T(s, n)) が値を持つことは, テキスト (impT+main) 上で T の停止性を証明することにより保証できる. よって, **命題 1**により, specP' $\ll$ (impT+main) である. $\equiv_{(\mathrm{impT+main+}i)} \supseteq \equiv_{\mathrm{SpecST}}$ であること証明は, 関数の適用に関する帰納法を用いて行う (**図 2**)



Base: $\frac{}{R(UI(SI),SI,N)}$

Ind: $\frac{R(SC,SI,N)\wedge\neg C_1(SC)\wedge\neg C_2(SC,N)}{R(UN(SC,N),SI,N)}$

Final1: $\frac{R(SC,SI,N)\wedge C_1(SC)}{Q(UE(SC,N),SI,N)}$

Final2: $\frac{R(SC,SI,N)\wedge\neg C_1(SC)\wedge C_2(SC,N)}{Q(UF(SC,N),SI,N)}$

For AX1: $C_1(s) \equiv \text{point}(s) =\mid \text{History } (s) \mid +1$

$C_2(s,n) \equiv \text{get}(\text{ point}(s), \text{History } (s)) = n$

$R(s,t,n) \equiv 1 \leq \text{point}(s) \leq\mid \text{History } (s) \mid +1$

$Q(s,t,n) \equiv \text{Out}(s) \neq 0 \supset n = \text{get}(\text{Out}(s),\text{History}(t))$

図 2: 証明図

Fig. 2 Proof of an Example Program

図 2は上述の帰納法における証明スキームを表している．状態遷移関数 T の実現がUI,UN,UF,UE を用いて図のような状態遷移で行われている．状態 SC で成り立つであろう不変式 Rを検証者が考案し，図 2にある 4 つの証明を行う．

例えば公理 AX1 の場合，R, C_1, C_2, Q としてそれぞれ図 2の下段の式を割り当てる．この結果とテキスト i における Set と History の関係式より，公理 AX1 が成り立つことが証明できる．

これらの証明を行うための証明支援システムが ASL システムでは用意されている．このシステム上で$\xi \approx \eta$であることを示すために s 上での項書換え，場合わけ，数学的決定問題への帰着，補題の追加，構造的帰納法等の方法を用いる[18]．なお，無矛盾性については，プログラムスタイル等より保証できる[18]．

例えば，まず (impT+main+i) のテキストを公理集合として読み込み，次いで各証明スキームの仮定に相当する式を補題追加機能を用いて追加しておく，そして結論部に相当する式ξをすでに入力されている公理集合を用いて順次書き換え，必要に応じて場合分けを行ったあと，整数上の恒真性判定機能などを用いて，$\xi \approx^{(\mathrm{SpecST+main})}_{(\mathrm{impT+main}+i)} \mathrm{TRUE}$ であることを結論づける．

不変式 Rの選択によっては，証明がうまくいかない場合もある．そのときのためにこのような不変式群の変更を容易にできるシステムも作成されている．

3.4 到達正当性条件

この節で到達正当性条件の記述スタイルの定義及びを到達正当性条件用いたテキストの詳細化の正しさの証明法について述べる．

3.4.1 到達正当性条件を用いた記述スタイルの定義

ASM 型記述の (ホ) のスタイルとして，

$\mathrm{valid}(s) \wedge \mathrm{Extern}_T(s,l) \wedge \mathrm{select}_T(s,l) \supset p_T(F_k(T(s,l)), F_{k'}(s), l) == \mathrm{TRUE}$ ..... (V.S)

のように記述する．これは「状態 s においてプログラムの外部入力 lが満たすべき条件 $\mathrm{Extern}_T(s,l)$ を満足し，かつ，lを引数とする状態遷移関数 Tを実行すべき ($\mathrm{select}_T(s,l)$

が真) ならば，T実行後の状態成分関数値 F_k，T実行前の状態成分関数値 $F_{k'}$，外部入力 lの間には述語 p_Tが成り立つ」ことを表している．ここで，Tが外部入力引数を持たなければ，$\mathrm{Extern}(s,l)$ の項はない．述語 p_Tは実際には，Tの前後の各状態成分関数 F_k間の関係を表す述語であるが，以後,「処理 Tの前後における関係を表す述語」であることを強調するために，$p_T(T(s,l),s,l)$ と状態成分関数を略して記述することもある．一方，各レベルにおいて,「初期状態 (Init で表す) でプログラムの外部入力 l'が，外部入力が正当であることを表す述語 vを満たすときに限り，初期状態の valid が真になる」よう公理 (Ax.1) で定義し，また,「初期状態から始まる実行系列において，各時点の外部入力 lが満たすべき条件 $\mathrm{Extern}_T(s,l)$ を満足しており，かつ，実際に到達した状態に対しては，valid が真になる」ように，プログラムの外部入力とプログラムに現われる各処理 Tについて公理 (Ax.2) で定義する．

$$\mathrm{valid}(\mathrm{Init}(l'))==v(l') \quad \cdots\cdots (\mathrm{Ax.1})$$

$$\mathrm{valid}(s)\wedge\mathrm{Extern}_T(s,l)\wedge\mathrm{select}_T(s,l) == \mathrm{valid}(T(s,l)) \quad \cdots\cdots (\mathrm{Ax.2})$$

先程と同様に Tが外部入力を持たなければ $\mathrm{Extern}_T(s,l)$ の項はない．仕様記述者は，$\mathrm{Extern}_T(s,l)$, $v(l')$ のみを基本述語等を用いて定義すれば良い．select_Tの定義は以降の節で述べるように，設計者の与えたこのレベルのループ関数によって自動的に定まる．

この記述方法では，外部入力に対する制限 Extern，v以外は処理の要求記述時に与えなくても良い．これにより，緒論で述べたことが利点となる．

3.4.2 制御状態を有限にするための記述制限

レジスタ付き有限状態機械と見なせるよう，各ループ関数 L_kの記述を以下の形に制限する．

$$\begin{aligned} L_k(s,l) == \ & \text{if } P_1(s,l) \text{ then } C_1 \\ & \text{else if } P_2(s,l) \text{ then } C_2 \\ & \quad \vdots \\ & \text{else } C_m \end{aligned} \quad \cdots\cdots (\mathrm{L.G})$$

ここで各 C_i は，ループ関数，状態遷移関数からなる状態遷移の系列で，ループ関数 L_pの出現は，最外に限られる．また，各 $P_i(s,l)$ は状態 s における状態成分関数と lに関する述

語である．

この形式にすると，制御部の状態数を有限にできる．以降の議論を簡単にするために，各 C_kを，$L_j(T_i(s,l))$ か，$T_i(s,l)$ の形に制限する．((L.G) の形式でループ関数が与えられたとき，ループ関数記号を追加して，この形のループ関数記述に直せるので一般性は失われない．) この制限を以下 (L.S) と表す．

第 kレベルの遷移の選択条件の自動導出

ここでは，第 kレベルの状態遷移関数 T_iを，第 $k+1$ レベルの各状態遷移関数 t_jと，制限 (L.S) を満たすループ関数 L_pを用いて展開する際，第 kレベルの T_iの適用前後の制御部の状態名 L_s, L_eと，第 kレベルでの T_iの選択条件 select_{T_i}および，(あれば) T_iの Extern_{T_i}が与えられたとき，(第 $k+1$ レベルの valid の定義の公理で必要となる) 第 $k+1$ レベルで新しく導入された状態名 や，各状態遷移 t_jの選択条件 select_{t_i}をどう与えるかについて述べる[3]．

但し，状態遷移関数 T_iに外部入力の引数 lがある場合，以下のように，引数 lはループ関数で展開される最初の状態遷移関数 t_0のみが引数とすることにする．すなわち，ループ関数は状態のみを引数とする．

$T_i(s,l) == L_p\ (t_0(s,l))$;

一般に L_pによる展開は，状態遷移図の一つの枝を 1 入口 1 出口の状態遷移グラフに置き換えることに相当する．第 $k+1$ レベルで新たに導入された状態に対して状態名を表す状態成分関数 name の値を以下のように定める．

t_0に関する name の公理は

[3]便宜上第 0 レベルの記述を考え，この記述は，初期状態 Init の指定と，一つの状態遷移 Tとその性質の記述，及び計算指定 $F(T(\text{Init}(\ldots),l))$ からなるとする．このレベルでは状態遷移 Tの前の状態 Init と，状態遷移 T後の状態 S_Fがあると見なせる．第 0 レベルの各状態における状態名は以下のように name(s) に保持できる．

name (Init(...)) == Init

valid(s)∧Extern(s,l) ⊃ name ($T(s,l)$) = if name (s) = Init then S_F == TRUE

また，このレベルの valid の定義は以下と思ってよい．

valid(Init(...))==v(...)

valid(s)∧Extern(s,l)∧ TRUE == valid($T(s,l)$)

第一レベルの記述は，Tを展開して得られる．このときの第一レベルの name や valid の公理は本文の展開則を用いて得られる．

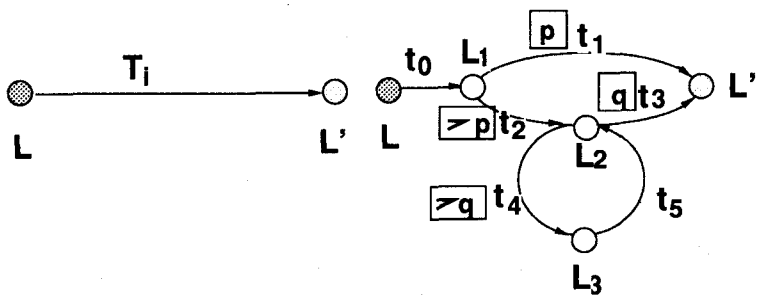


図 3: 展開の例

Fig. 3 Example of Refinement

$\text{valid}(s) \wedge \text{Extern}_{t_0}(s,l) \wedge \text{select}_{t_0}(s,l) \supset$

$$\text{name}\ (t_0(s,l)) = \text{if name}(s){=}L_s \text{ then } L_p == \text{TRUE} \quad \ldots\ldots\ldots\ldots\ldots\ldots (\text{N.R0})$$

となる[4]．ループ関数 L_pの展開で用いられる各状態遷移 t_iについては，

$\text{valid}(s) \wedge \text{select}_{t_i}(s) \supset$

$$\text{name}\ (t_i(s)) = \text{if name}(s) = L_p \text{ then } L == \text{TRUE} \quad \ldots\ldots\ldots\ldots\ldots\ldots (\text{N.R})$$

とする．ここで L は以下のように与える．各状態遷移 t_iに対して，この展開において，t_iの出現は，$L_j(t_i(s))$ の形か，$t_i(s)$ のように単独であるかいずれかである．前者の場合 L は L_j とする．後者の場合は L は T_iの適用後の状態名である L_eとする．

t_0に関する valid の公理で使われる select_{t_0}, Extern_{t_0}の定義は，T_iの select_{T_i}, Extern_{T_i} と同じものにする (S.E)，(図 **3**).

一方，t_0 以外の一般の状態遷移関数 t のための select_t は以下のように定義する．

$$\text{select}_t(s) == \text{if name}(s){=}L_i \text{ then } P_i'(s) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (\text{D.S})$$

ここで，$P_i'(s)$ は，(L_iなる状態において，) ループ関数 L_iが t を選択するときの実行条件である．

第 kレベルや，第 $k+1$ レベルにおいて，(同じ名前の状態遷移)T_i, t_jが異なった複数の状態で適用されることがある．異なった出現の T_iごとに与えられる状態遷移の部分グラフの形は (ループ関数は同一なため) 同型である．状態名については，各 T_iごと異なった状態

[4]if p then g は，p が真と合同であるときのみ，g と合同である．

表 4: valid の公理

(vi):valid(s) $\wedge$ select$_{T_i}$(s,n) $\wedge$ Extern$_{T_i}$(s,n) == valid(T_i(s,n))

(v0):valid(s) $\wedge$ select$_{t_0}$(s,n) $\wedge$ Extern$_{t_0}$(s,n) == valid(t_0(s,n))

(v1):valid(s) $\wedge$ select$_{t_1}$(s) == valid(t_1(s))

(v2):valid(s) $\wedge$ select$_{t_2}$(s) == valid(t_2(s))

(v3):valid(s) $\wedge$ select$_{t_3}$(s) == valid(t_3(s))

(v4):valid(s) $\wedge$ select$_{t_4}$(s) == valid(t_4(s))

(v5):valid(s) $\wedge$ select$_{t_5}$(s) == valid(t_5(s))

(L0): T_i(s,n) == L_1(t_0(s,n)); (L1): L_1(s) == if p(s) then t_1(s) else L_2(t_2(s));

(L2): L_2(s) == if q(s) then t_3(s) else L_3(t_4(s)); (L3): L_3(s) == L_2(t_5(s));

(e0):Extern$_{t_0}$(s,n) == Extern$_{T_i}$(s,n);

(s0):select$_{t_0}$(s,n) == select$_{T_i}$(s,n);

(s1):select$_{t_1}$(s) == if name(s)=L_1 then $p(s)$;

(s2):select$_{t_2}$(s) == if name(s)=L_1 then $\neg p(s)$;

(s3):select$_{t_3}$(s) == if name(s)=L_2 then $q(s)$;

(s4):select$_{t_4}$(s) == if name(s)=L_1 then $\neg q(s)$;

(s5):select$_{t_5}$(s) == name(s)=L_3;

(n0):valid(s) $\wedge$ select$_{t_0}$(s,n) $\wedge$ Extern$_{t_0}$(s,n) $\supset$

name(t_0(s,n)) = if name(s)=L then L_1 == TRUE;

(n1):valid(s) $\wedge$ select$_{t_1}$(s) $\supset$ name(t_1(s)) = if name(s)=L_1 then L' == TRUE;

(n2):valid(s) $\wedge$ select$_{t_2}$(s) $\supset$ name(t_2(s)) = if name(s)=L_1 then L_2 == TRUE;

(n3):valid(s) $\wedge$ select$_{t_3}$(s) $\supset$ name(t_3(s)) = if name(s)=L_2 then L' == TRUE;

(n4):valid(s) $\wedge$ select$_{t_4}$(s) $\supset$ name(t_4(s)) = if name(s)=L_2 then L_3 == TRUE;

(n5):valid(s) $\wedge$ select$_{t_5}$(s) $\supset$ name(t_5(s)) = if name(s)=L_3 then L_2 == TRUE;

名を与えるために，第 $k+1$ レベルの (N.R) の name を与える公理において，ループ関数名 L_j と (第 n 番目の出現を意味する)n の組を name に与える等の変更をすればよい．またこのとき，T_iにつき (N.R0) や (N.R) は複数の公理になる．そのときは，if then else のカスケードを用いて，一つの公理にまとめる．これは (D.S) についても同様である．

展開の例を (**図 3**, **表 4**) にあげておく．ここでは，T_i を公理 (L0), (L1), (L2), (L3) のループ関数で展開しており，その場合に対応する状態機械の制御部は図の通りである．下位のテキストの状態遷移 $t_0, t_1, \ldots, t_5$の valid の公理は (v0), ..., (v5) であり，また各 select の定義は，(s0),..., (s5) で表される．また 各処理の name.に関する定義式は公理 (n0),...,(n5) に与えられている．

3.4.3 到達正当性条件を含むテキストにおける詳細化の正しさの定義

このように記述されたテキスト A, B に対して，**定義 7**のもとで Bが A の正しい詳細化であるためには，A にあって Bに存在しない公理が，B上でも定理として成り立てばよい．これらの公理には, (I) テキスト A の状態名 name に関する公理，及び select の公理，(II) 状態遷移関数 T_iに関する valid の定義の公理，(III) テキスト Bで展開される T_iについて，テキスト A で性質記述 (ホ) として使われる公理．がある．

(II) の公理群は先程の詳細化によって機械的に導出した下位の記述のもとで定理として成り立たない．実際，項の組 $\langle$ valid(s)$\wedge$ select $\cdots$, valid($T(s)$)$\rangle$ は，下位レベルにおいて $T(s)$ の値が定まらない場合に[5] ，合同関係が満たされない．よって定理として成り立たない．しかし，実用的見地からは，下位レベルにおいては，例えば Extern を満たさない不正な入力対しては Tが停止しないような実現も許されるべきである．そこで，valid を含むテキスト間の詳細化の正しさを次の**定義 9**で定義する．この定義では，上位の valid に対する要求を含意 ($\supset$) を用いて，valid(s)$\wedge$ select $\cdots \supset$ valid($T(s)$) が成り立つこととしている．

[定義 9] [到達正当性条件を含むテキスト間の詳細化の正しさ]

順序機械型プログラムを **3.2節**で述べた詳細化方法で詳細化し，ループ関数の記述クラ

[5]すなわち, 下位レベルのテキスト上の状態遷移関数で, $T(s)$ を展開したときに有限長で表せないとき, (以降では状態遷移の値が定まらないというのをこの意味で用いる.)

スは **3.2節**の (L.S) を満たし，valid 等の定義は **3.4.2節**に従うものとする．　(記述条件 (D.C))

このようなテキスト A, Bに対し，A の valid の定義式 (Ax.2) について，

$\text{valid}(s) \wedge \text{Extern}(s,l) \wedge \text{select}_T(s,l) \supset \text{valid}(T(s,l)) == \text{TRUE}$ ..... (Ax.2')

と読み変えたテキストをテキスト A'とし，テキスト Bが A'の正しい詳細化 (**定義 7**) であれば，Bが A の正しい詳細化であるといい，この関係を $A \lhd B$と表す．　□

関係$\lhd$は推移律が成り立つ．この性質は，段階的詳細化では自然に要求される性質である．

3.4.4　到達正当性条件を含むテキストにおける詳細化の正しさの証明法

記述条件 (D.C) を満たすテキスト A, Bについて，$A \lhd B$を満たすためには，以下が成り立てば良い．

前述の (I) が B上で定理として成り立つこと．これは，name の定義 (N.R) と，記述スタイルの制限 (L.S) より自明である．(II) については (Ax. 2') が B上で成立つこと．これは**命題 2**で保証する．(III) は後述する．

[命題 2]

記述条件 (D.C) をみたすテキスト A, Bに対し，テキスト A の性質記述 (ホ) で対象となった各状態遷移関数 T_iについて $\text{valid}(s) \wedge \text{select}_{T_i}(s,n) \wedge \text{Extern}_{T_i}(s,n) \equiv_B \text{TRUE}$ を満たす任意の s, n への A 上の許される代入ρに対して$\rho(T_i(s,n))$ が B上で値を持てば[6]，$\text{valid}(s) \wedge \text{select}_{T_i}(s,n) \wedge \text{Extern}_{T_i}(s,n) \supset \text{valid}(T_i(s,n)) \approx_B^A \text{TRUE}$ である．　□

証明: いま，議論の簡単のため，テキスト A 上での T_iの出現を一回にしておく．また，テキスト A 上での T_iを行った直後の状態名を L_eと仮定する．

$\text{valid}(S) \wedge \text{select}_{T_i}(S,N) \wedge \text{Extern}_{T_i}(S,N) \equiv_B \text{TRUE}$ でないことを仮定すると，この定義より題意が成り立つ．一方 B 上で TRUE と合同であるとき ($\cdots$ (SP)) に，$\text{valid}(T_i(S,N)) \equiv_B \text{TRUE}$ が成り立つことを B 上のループ関数による展開の回数 n に関する帰納法で証明する．

[6]すなわち，T_iの B上の遷移による展開が有限長であれば

$n=0$ のとき，記述条件 (D.C) より $\mathrm{valid}(T_i(S,N)) \equiv_B \mathrm{valid}(L_k(t_0(S,N)))$ であり，このとき，name の定義 (N.R0) より，$\mathrm{name}(t_0(S,N)) \equiv_B L_k(=L_e)$ となる．また，t_0の select, Extern の定め方 (S.E) と仮定 (SP) より $\mathrm{valid}(t_0(S,N)) \equiv_B \mathrm{TRUE}$.

さて，一般に $T_i(S,N) \equiv_B L_i(t_n(\cdots t_0(S,N)\cdots))$ であり，$\mathrm{valid}(t_n(\cdots t_0(S,N)\cdots)) \equiv_B \mathrm{TRUE}$ かつ，$\mathrm{name}(t_n(\cdots t_0(S,N)\cdots)) \equiv_B L_i$が成り立つとする (RAS)．今，((L.S) を満たす) ループ関数 L_iを展開する．実行条件式 $P(t_n(\cdots t_0(S,N)\cdots))$ への真偽割当ては有限である．ある真偽割当てで，$T_i(S,N) \equiv_B L_j(t_{n+1}(t_n(\cdots t_0(S,N)\cdots)))$ であれば，select の定義 (D.S) と，帰納法の仮定 (RAS) の name の値より，$\mathrm{select}_{t_{n+1}}$が真．このとき (RAS) の前半と，Ax.2 より，$\mathrm{valid}(t_{n+1}(t_n(\cdots t_0(S,N)\cdots))) \equiv_B$TRUE である．また，$\mathrm{name}(t_{n+1}(t_n(\cdots t_0(S,N)\cdots)))$ の値は，帰納法の仮定 (RAS) の name の与え方と，(N.R) より L_jとなる．また，これは，任意の真偽割当てに対し成立する．

$\mathrm{valid}(s) \wedge \mathrm{select}_{T_i}(s,n) \wedge \mathrm{Extern}_{T_i}(s,n) \equiv_B \mathrm{TRUE}$ を満たす任意の s,n への A 上の許される代入ρに対して$\rho(T_i(s,n))$ のテキスト B上での状態遷移の展開が有限である仮定より，$\mathrm{valid}(T_i(s,n)) \approx_B^A \mathrm{TRUE}$ となる． □

最後に，(III) について述べる．valid の入った式のまま議論できないので，テキスト A 上で状態 s に関して，valid(s) $\supset P(s)$ を満たすと予想される内部状態成分関数間の具体的な言明 $P(s)$ を検証者が考案し，これを用いて証明する方法を用いる．

一般に状態遷移の性質の記述は，本章の最初で述べたように (V.S) の形式で書かれている．この形式においてつ以降の部分を (処理の) 後置条件と呼ぶ．また，$\mathrm{select}_{T_i}(s) \supset \mathrm{select}'_{T_i}(s)$ が成り立ち，かつ容易に検証者が考案できる select'_{T_i} (例えば，実行指定の if 文の条件判定式等) を以降で用い，$\mathrm{select}_{T_i}(s) \supset \mathrm{select}'_{T_i}(s)$ が成り立つ性質を適時用いる．

一般にテキスト A(あるいは B) は，(イ) **初期化**，(ロ) **状態遷移の定義**の記述，(ハ) **ループ関数**，(ニ) **計算指定**及び，(ホ) **状態遷移の性質**の記述からなる．テキスト A'(あるいは，B') を，テキスト A(あるいは B) の (ホ) の各公理をそれぞれ (前提条件を取って) 後置条件が真であるという公理に置換えて A(あるいは B) から得られるテキストとする．

[方法 2]

次の (i)の後，(ii)，(iii) を証明して，テキスト A 上で要求性質が記述された各状態遷移

T_iについて，valid(s)$\wedge$ select$_{T_i}(s,l)\wedge$ Extern$_{T_i}(s,l) \supset p_{T_i}(T_i(s,l),s,l) \approx_B^A$ TRUE を結論づける．

(i). 検証者が，テキスト A 上で valid(s) $\supset P_B(s)$ を満たすと思われる言明 $P_B(s)$ を考案する．

(ii). (i)の言明 $P_B(s)$ に対し，次の (a)，(b) が成り立つこと．

(a) $v(l') \supset P_B(\text{init}(l')) \approx_{A'}^{A}$TRUE

(b) テキスト A上の各状態遷移T_iに対し，$P_B(S)\wedge$select$'_{T_i}(S,L)\wedge$Extern$_{T_i}(S,L) \equiv_{A'}$TRUE を仮定して，$P_B(T_i(S,L)) \equiv_{A'}$ TRUE であること．

(iii). テキスト A 上の各状態遷移 T_iに対し，テキスト B'上で，$P_B(S) \wedge$ select$'_{T_i}(S,L) \wedge$ Extern$_{T_i}(S,L) \equiv_{B'}$TRUE を仮定して，$p_{T_i}(T_i(S,L),S,L) \equiv_{B'}$TRUE であること．□

[命題 3]

方法 2により，テキスト A 上で要求性質が記述された各状態遷移 T_iについて，valid(s)$\wedge$ select$_{T_i}(s,l)\wedge$ Extern$_{T_i}(s,l) \supset p_{T_i}(T_i(s,l),s,l) \approx_B^A$ TRUE が成り立つ．□

方法 2の利点は次の 2 点である．テキスト A,Bより簡明な A',B'上で議論できることと，言明 P_Bの不変性の証明が，A'上の大きな状態の単位で行えることである．

命題 3の証明: まず，テキスト A 上の状態を表す項 Sの構成に関する帰納法を用いて，

「 valid(S) $\equiv_B$ TRUE なる Sについて，$P_B(S) \equiv_B$TRUE が成り立つこと」$\cdots(*)$ を示す．

基底段階: valid(Init(l')) $\supset P_B$(Init(l')) $\equiv_B$ TRUE が成り立つことは，(D.C) 及び，**方法 2** (ii)(a) より，自明．

帰納段階: valid(SC) $\equiv_B$ TRUE なる状態 SC について $P_B(SC) \equiv_B$ TRUE であることを帰納法の仮定とする．

今，valid($T_i(SC,L)$) $\equiv_B$ TRUE を仮定する．このとき，テキスト B上の valid の公理 (Ax. 2) より valid(SC)$\wedge$ select$_{T_i}(SC,L)\wedge$ Extern$_{T_i}(SC,L) \equiv_B$ TRUE である．よって，このことと，帰納法の仮定，**補題 2**，及び**方法 2**(iii)より，

$p_{T_i}(T_i(SC,L),SC,L) \equiv_B$ TRUE (H1)

ここで，状態SCに関して，T_iの到達正当条件部が真であること，と (H1) より，A のT_iの処理の性質の公理は，状態SCにおいては，B上で合同関係を持つ．よって，**方法 2**(ii)(b) とあわせて，$P_B(T_i(SC,L)) \equiv_B$TRUE が成り立つ．

以上より，「valid$(T_i(SC,L)) \equiv_B$ TRUE なる状態 $T_i(SC,L)$ について $P_B(T_i(SC,L)) \equiv_B$ TRUE」が成り立つ．

次に，任意のT_iについて，「valid$(s) \wedge$ select$_{T_i}(s,L) \wedge$ Extern$_{T_i}(s,L) \supset p_{T_i}(T_i(s,L),s,L) \approx_B^A$ TRUE が成り立つこと」$\cdots(**)$ を示す．

A 上で定義される任意の状態SCについて，$(**)$ であることは，上述の帰納段階の証明の過程で証明されている．実際，帰納段階の証明において，状態SCにおいて，valid$(SC) \wedge$ select$_{T_i}(SC,L) \wedge$ Extern$_{T_i}(SC,L) \equiv_B$ TRUE を仮定すると，$p_{T_i}(T_i(SC,L),SC,L) \equiv_B$ が導出される．一方，偽の場合を仮定すると $(**)$ のつの前半が偽となり，$(**)$ が成り立つ．すなわち題意が結論される． □

[補題 2]

valid$(SC) \wedge$ select$_{T_i}(SC,L) \wedge$ Extern$_{T_i}(SC,L) \equiv_B$ TRUE ならば，**方法 2**(iii)を B'上で行っても，$p_{T_i}(T_i(SC,L),SC,L) \equiv_B$ TRUE が結論できる． □

証明の方針: テキスト Bの (Ax. 2) 等を用いて証明する． □

3.4.5 前提条件を陽に記述する方法との比較

提案した手法と従来の (valid の代わりに具体的な述語を直接記述する) 手法を証明の論法の観点から比較する．従来の手法では，前提条件を (選択条件を含め) 具体的に記述している (例えば文献[36]の処理の性質など)．従来手法では，T_iをテキスト B 上で t_i等を用いて詳細化しているとき，(1)T_iの前提条件 Pre から結論される述語で各 t_iの具体的な前提条件を真にする，状態に関する言明 Rを検証者が考案し，(2) 言明 Rが各 t_iの適用の前の状態で t_iの前提条件を導出すること，(3) 言明 Rが各 t_iの適用の後の状態で不変式として成り立つこと，(4) 各 t_iの要求記述 ((2),(3) より実際は後置条件のみでよい)，R と詳細化のループ関数等を用いてテキスト B上で T_iの要求が真になること，を証明する．**方法 2**の (iii)では，

(1) における T_iの前提条件 Pre が P_Bに対応する．Rに対応する不変式も (必要があれば) 同様に考案する必要がある．但し，(2) の作業は，本手法では到達正当性条件が**補題 2** より自動的に成り立つので不要である．(3)，(4) に相当する作業は本手法の (iii)でも必要である．従来の手法では，テキスト A の各状態遷移の展開ごと独立に証明を行う．一方，**方法 2**では (ii)のような作業が必要である．**方法 2**の (ii)は，従来の「Pre が確かに T_iの実行前の状態で成り立つことを保証すること」に相当する．従来でも，そのことを保証したければ証明する必要がある．なお，テキスト A が，より上位に対する実現であること証明する際には，当然 **方法 2**の (ii)に相当する (2)，(3) を行うし，テキスト Bの実現を考えるときには，(2)，(3) に相当する**方法 2**の (ii) を行うので，行うことはレベルがずれているだけである．また，従来の記述法では，状態遷移の独立性 (前提条件を満たすところでは，無条件にその状態遷移を用いれること) があるが，本記述法でも**方法 2** の (ii) で得られた P_B，Extern_tと select'_tを満たす状態では，t を用いることができ，遜色がない．また本手法で，効率，エラー処理等を考慮して以降の詳細化を行う際にも，この P_B，Extern_tと select'_tを前提条件とみなして，行えばよい．ここで，select' は，前節で定義した select'である．

3.5 実用例題への適用例

前節で導入した諸概念の在庫管理問題への適用例をこの節で述べる．

3.5.1 在庫管理問題

ここで在庫管理問題について簡単に概説する (図 4参照)．

(1). (どの商品を何個出力するかを記載した)「出庫依頼書」を受け付けたとき，在庫があれば,(どの商品をどのコンテナから何個ずつ出庫するか，また，その出庫によりそのコンテナが空になるかどうか等を記載した)「出庫指示書」を出力する．在庫がなければ「出庫不可連絡」を出力する．

(2). (そのコンテナにどの商品を何個格納しているかを記載した)「積荷票」を受け付けたとき，過去に在庫不足で出庫できなかった「出庫依頼書」の中で，その入庫により出庫出来るものが生じれば，それらに対してそれぞれ「出庫指示書」を出力する．

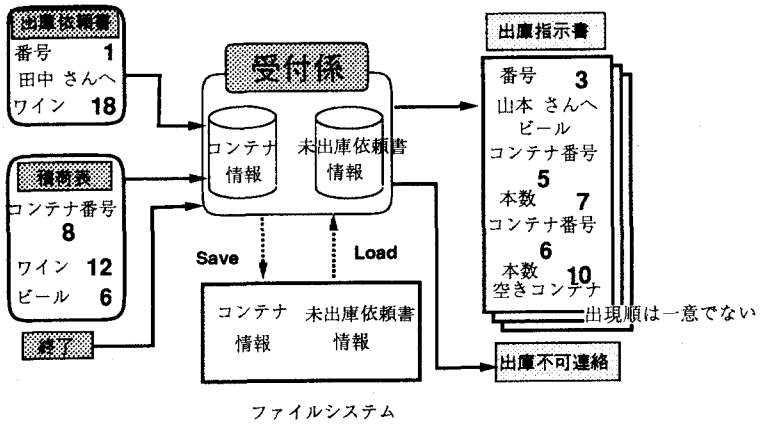


図 4: 在庫管理

Fig. 4 Stock Management

(3). 「終了」を受け付けるまで 上記 (1),(2) を繰り返す.「終了」を受け付けたとき, 在庫情報等の内部情報を適当なファイルに保存し, 再起動時にはこのファイルから情報を取り戻す.

3.5.2 在庫管理の記述のための基本データ型と基本関数

ここで在庫管理のプログラム, 仕様記述に用いる基本データ型と基本関数について簡単に述べる. 在庫管理の入力データである積荷表と出庫依頼書, 出力データである出庫指示書 (以下単に依頼書, 指示書とも記述する) のデータ型は, 一般的な, 文字列, 整数及び 両方向リストなどの組として定義される. 両方向リストはポインタを内部で持っており, 抽象リストと呼ぶことにする. また, 各抽象リストの要素をセルと呼ぶことにする. セルのデータ型が異なっていていも共通のリスト演算子が用いられるものとする. 抽象リストの基本演算として次のようなものがある. : **新リスト, トップ, ボトム, 前進, 後退, 削除+, 削除−, 挿入, 参照, 参照_i, 変更_i, 先頭, 後尾** . 詳細は文献[36, 43, 44, 45, 46]を参照.

3.5.3 第 1 レベルの記述例 (被拡張射影テキスト)

在庫管理を 5 レベルで詳細化した. この詳細設計の概念図を図 5に表す.

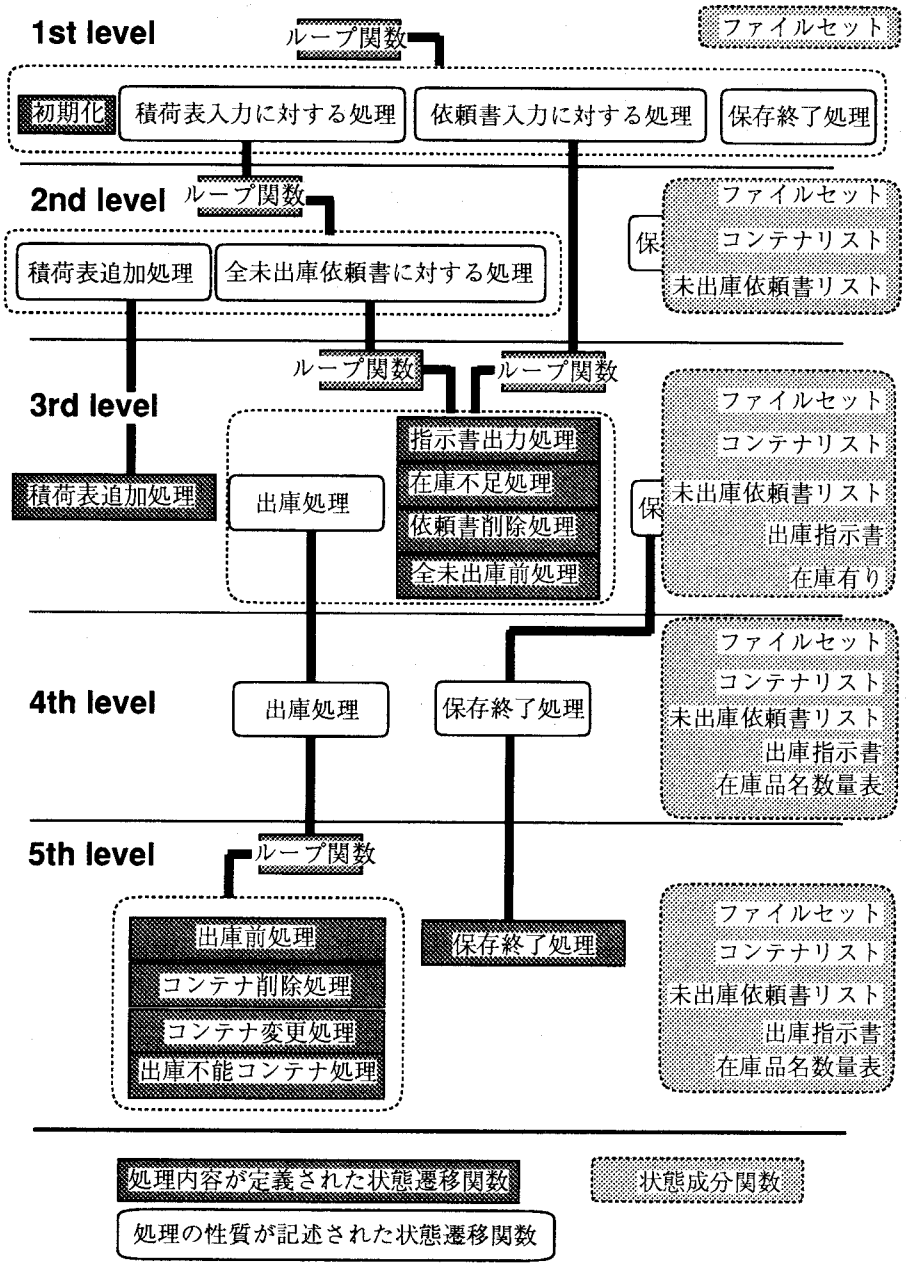


図 5: 在庫管理の階層的設計

Fig. 5 Stepwise Refinements of Stock Management

第1レベルで考えている状態遷移関数は**依頼書入力に対する処理**, **積荷表入力に対する処理**, **保存終了処理**の3つである. これらの処理の要求記述を行なう際, 例えばどのコンテナから処理していくか, あるいはどの未出庫依頼書から処理していくか, などの要求に自由度を持たせたい. そこでまず, これらの処理の要求記述を標準的と思われる状態成分関数を用いて行なう. この記述を行なったテキストを $TL1$ とする. そして, テキスト $TL1$ からこれらの処理に関して関心ある出力 (出力用ファイル, 情報保存用ファイル) だけを拡張射影で取り出す. これに関しては **3.5.4節**で述べる. ここではテキスト $TL1$ について述べる. 状態成分関数として, コンテナリスト (S), 未出庫依頼書リスト (S), ファイルセット (S) を用意する. それぞれ, 状態 S での在庫情報, 未出庫依頼書情報, ファイルの集合を表すのに使用する. コンテナリスト, 未出庫依頼書リストのデータ型は抽象リストとする.

ここでは, 特に**積荷表入力に対する処理**の要求記述について述べる. この処理は, (外部入力である) 積荷表により, (以前の在庫情報と合わせて) 出庫できるようになった (出庫できず残っていた依頼書である) 未出庫依頼書をすべて出庫する処理である. **表 5**中の**要求 A ～ C** 等をどのように表現するかが重要な点である. ここでは**積荷表入力に対する処理**に対して**性質 1 ～ 3** を要求することとする.

これらの性質に対応する 各公理は前述の valid を用いて満たすべき性質がどうあるべきか, という形で記述されている (**表 6**参照).

なお "依頼書や, 積荷表に現れる本数は正である" や, "番号は内部で記憶しているコンテナ番号や, 依頼書番号と異なる" という, 外部入力に関する条件は, 積荷表正当として与える.

公理 I1 が **性質 1** a) に相当する. 関数 依頼書対応は **性質 1** b) の内容に対応することを表す述語である. 公理 I2, I3 , SS6 がそれぞれ**性質 3** a), **性質 3** b), **性質 1** c) に相当する. 公理 SS6 では (関数ボトムでポインタを後尾にセットすることにより) 出庫指示書リスト全体に対して関数 **指示書リスト正当**が成り立つことを記述している. 関数 **指示書リスト正当**は別テキストで, リストの先頭からポインタまでのすべての位置の指示書に対して, 述語 **指示書正当** (**表 5**の (S6a), (S6b)) が成立することと (再帰的に) 定義されている. その他**表 5** のコンテナリストの性質に相当することも記述する (ここでは省略, 同様の記述は以下に述べる**依頼書入力に対する処理**の要求性質の記述にある.).

表 5: 性質記述すべき内容

要求 A 未出庫依頼書情報の中から出庫できるものはすべて出庫すること
要求 B 依頼書を選ぶ順番を陽に指定しないこと
要求 C 作成された出庫指示書リスト中の任意の指示書がその出現位置で正しい記載内容をもつこと
性質 1 その処理で作成される出庫指示書のリストを想定し, そのリストに対し,
a) 個々の指示書は 処理以前の未出庫依頼書リストに該当する依頼書があり, かつ
b) 該当する依頼書に対応した内容であること,
c) 出庫指示書のリストが, **要求 C** すなわち以下の**指示書正当**を満たすこと.
性質 2 処理後のコンテナリストは, 出庫指示書のリストに書かれた分出庫されていること.
性質 3 処理後の未出庫依頼書情報は,
a) 出庫指示書のリストに該当する依頼書の分を処理以前の未出庫依頼書情報から取り除いたものであること.
b) 処理後の在庫情報に対して出庫できるものがないこと.

性質 1 b) の定義は以下の通り.

(S1) 指示書の番号が依頼書の番号に一致すること.
(S2) 指示書の送り先が依頼書の送り先に一致すること.
(S3) 指示書の品名が依頼書の品名に一致すること.
(S4) 指示書中のコンテナの記述は一度に限る (同じコンテナに関する記述が 2 回以上現れない) こと.
(S5) 指示書に書かれた本数の総数と依頼書の本数が一致すること.

指示書正当の内容

(S6a) 指示書に書かれた本数は正で, その本数はそれまでの仮想出庫量を差し引いても当該コンテナに存在すること.
(S6b) それまでの仮想出庫量に加え, 指示書に書かれた本数だけ当該コンテナから出庫するとそのコンテナが空になるときそのときのみ, 指示書に空きコンテナ指示が与えられること.

コンテナリストの性質

(C1) 出庫前のコンテナリストにあるコンテナで, かつ出庫指示書に現れかつ, 空きコンテナ指示がないコンテナに関して, 次の (a),(b) を満たすこと.
(a) そのコンテナ番号は, 出庫後のコンテナリストにも, ただ一つあること.
(b) その番号のコンテナに関して, 次の i,ii を満たすこと.
i 出庫品目は, 出庫量だけ減少していること.
ii 出庫品目以外の品目に関して, 出庫前後で品目及び数量に変化がないこと.
(C2) 出庫前のコンテナリストにあるコンテナで, かつ出庫指示書に現れないコンテナに関して, 次の (a),(b) を満たすこと.
(a) そのコンテナ番号は, 出庫後のコンテナリストにも, ただ一つあること.
(b) その番号のコンテナに関して出庫前後で品目及び数量に変化がないこと.
(C3) 出庫前のコンテナリストにあるコンテナで, かつ出庫指示書に空きコンテナ指示があれば, 当該コンテナはコンテナリストから外されること.
(C4) 出庫前のコンテナリストにないコンテナは出庫後のコンテナリストに現れないこと.

表 6: 積荷表入力に対する処理の要求記述 (一部)

```
valid(S 積荷表追加処理 (積荷表)) ==
     vaild(S) and and Select 積荷 (積荷表, S) and 積荷表正当 (積荷表, S) ;
積荷表正当 (積荷表, S) == 各本数 (積荷表)>0 and
            Disjoint_id(トップ (コンテナリスト (S)), 積荷表) ;
define VALID := 'vaild(S 積荷表入力に対する処理 (積荷表))' ;
define 処理後指示書リスト :=
 '差リスト (ファイルシステム (S), ファイルシステム (S 積荷表入力に対する処理 (積荷表)))';
define 処理後未出庫依頼書リスト := '未出庫依頼書リスト (S 積荷表入力に対する処理 (積荷表))' ;
I1: VALID imply( Memberof(id, 処理後指示書リスト) ⊃
    Memberof(id, 未出庫依頼リスト (S)) and
      依頼書対応 ( 参照 (id, 処理後未出庫依頼リスト),
                  参照 (id, 処理後指示書リスト))) == TRUE ;
I2: VALID imply 集合 (処理後未出庫依頼リスト) =
    集合 (未出庫依頼リスト (S)) - 依頼書集合 (処理後指示書リスト) == TRUE ;
I3: VALID imply(Memberof(id, 処理後未出庫依頼リスト)
 ⊃ not 出庫可能 ( 参照 (id, 処理後未出庫依頼リスト),
          コンテナリスト (S 積荷表入力に対する処理 (積荷表))) == TRUE ;
                                    ⋮
SS6: VALID imply
指示書リスト正当 (コンテナリスト (S), 積荷表, ボトム (処理後指示書リスト),CargoID) == TRUE ;
```

関数 **依頼書入力に対する処理**の要求性質は, 例えば, 在庫がある場合に**表 5**の (S1)~(S6) 等に相当すること等や, コンテナリストの性質 (C1)~(C4) に相当すること, また 在庫がない場合に未出庫依頼情報に引数の依頼書の情報が加わること, 該当依頼書の出庫不可連絡が出力されること等であり, これに対応する公理を記述する (記述例は**表 7**).

関数 **保存終了**の満たすべき性質として, どのファイルにどの内部情報が保存されるべきかを記述する.

3.5.4 第 1 レベルの記述例 (拡張射影による抽出)

第 1 レベル記述テキスト本体では, 上述の 3 つの状態遷移関数については, 入力と処理後の (ファイルセットの中で興味ある) ファイルの値に関する合同関係だけを拡張射影を用いて例えば以下のように 取り出す記述を行なう.

依頼書入力に対する処理, 積荷表入力に対する処理, 保存終了をそれぞれ T_1, T_2, T_3 と略記する.

出力ファイル (ファイルセット (S T_1(irai))) == file @@**text** TL1;

出力ファイル (ファイルセット (S T_2(tumini))) == file @@**text** TL1;

表 7: 依頼書入力に対する処理の要求記述

```
define 出庫可 := ' 在庫有り (S, pr'3(依頼書), pr'4(依頼書))' ;
define 出庫可条件   := 'valid(S 依頼書入力に対する処理 (依頼書)) and 出庫可' ;
define 出庫不可条件:= 'valid(S 依頼書入力に対する処理 (依頼書)) and not 出庫
可' ;
define 指示書 := ' トップ指示書 (ファイルセット (S),
          ファイルセット (S 依頼書入力に対する処理 (依頼書)))'

  (* ファイルセットに関する性質 *)
O1: 出庫可条件 imply  番号   (指示書)=番号  (依頼書) == TRUE ;
O2: 出庫可条件 imply  送り先 (指示書)=送り先 (依頼書) == TRUE ;
O3: 出庫可条件 imply  品名   (指示書)=品名  (依頼書) == TRUE ;
O4: 出庫可条件 imply  Unique'id(指示書)      == TRUE ;
O5: 出庫可条件 imply
   総本数'指示書 (指示書)=本数 (依頼書) == TRUE ;
O6: 出庫可条件 imply
( Memberof(CargoID, 指示書) ⊃ (
    本数'在庫. 番号. 品名 (コンテナリスト (S), CargoID, 品名 (指示書))
    ≧ 本数'指示書. 番号 (指示書, CargoID) > 0 and
    搬出コンテナ (指示書, CargoID) iff
    本数'在庫. 番号 (コンテナリスト (S), CargoID)=
    総本数'指示書. 番号 (指示書, CargoID)
) ) ==TRUE ;
O7: 出庫不可条件 imply
  ファイルセット (S 依頼書入力に対する処理 (依頼書))=
  ファイルセット (S) Apend'string(出力ファイル,
                連絡文字列化 (指示書 (依頼書))) == TRUE ;

  (* コンテナリストの性質 *)
define 処理後コンテナリスト :=
  ' コンテナリスト ( S 依頼書入力に対する処理 ( 依頼書 ) )' ;
define '搬出指示' := ' 搬出指示コンテナ (指示書, CargoID)' ;
define '在庫減少量' :=
  ' 本数'在庫. 番号. 品名 (コンテナリスト (S), CargoID, 品名 (指示書))
  - 仮想数量 ( 品名 (指示書),
          表'在庫. 番号 (処理後コンテナリスト, CargoID) )' ;

C1a: 出庫可条件 imply
( Memberof(CargoID , 指示書) and not 搬出指示
 and Memberof(CargoID, コンテナリスト (S)) ⊃
    Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C1bi: 出庫可条件 imply
( Memberof(CargoID, 指示書) and not 搬出指示 and
  and Memberof(CargoID, コンテナリスト (S)) ⊃
  Uniq'id(処理後コンテナリスト) and
  在庫減少量 = 本数'指示書. 番号 (指示書, CargoID) )  == TRUE ;
C1bii: 出庫可条件 imply
( Memberof(CargoID, 指示書) and not 搬出指示
 and Memberof(CargoID, コンテナリスト (S))
 and not 品名 (指示書) = hin ⊃
 Uniq'id(処理後コンテナリスト) and
 コンテナ不変 (hin,
    表'在庫. 番号 (処理後コンテナリスト, CargoID),
    表'在庫. 番号 (コンテナリスト (S), CargoID)) )== TRUE;
C2a: 出庫可条件 imply
( not Memberof(CargoID, 指示書)
  and Memberof(CargoID, コンテナリスト (S)) ⊃
  Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C2b: 出庫可条件 imply
( not Memberof(CargoID , 指示書)
  and Memberof(CargoID, コンテナリスト (S)) ⊃
  Uniq'id(処理後コンテナリスト) and
  コンテナ不変 (hin,
     表'在庫. 番号 (処理後コンテナリスト, CargoID),
     表'在庫. 番号 (コンテナリスト (S), CargoID)) )== TRUE;
C3: 出庫可条件 imply
( Memberof(CargoID , 指示書) and  搬出指示
 and Memberof(CargoID, コンテナリスト (S)) ⊃
 Uniq( Set'Top'z( SYOKKOCONT ) ) and
 not Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C4: 出庫可条件 imply
( not Memberof(CargoID, コンテナリスト (S)) ⊃
  Uniq( Set'Top'z( SYOKKOCONT ) ) and
  not Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C5: 出庫不可条件 imply
処理後コンテナリスト=コンテナリスト (S) == TRUE;

  (* 未出庫依頼書リストに関する性質 *)
define 処理後未出庫依頼書リスト :=
  ' 未出庫依頼書リスト (S 依頼書入力に対する処理 (依頼書))'

I1: 出庫可条件 imply 処理後未出庫依頼リスト =
                  未出庫依頼リスト (S) ==TRUE ;
I2: 出庫不可条件 imply 処理後未出庫依頼リスト =
  後尾追加 (未出庫依頼リスト (S), 依頼書 ) ==TRUE ;

  (* valid の定義 *)
valid(S 依頼書入力に対する処理 (依頼書)) ==
    vaild(S) and Select 依頼書入力 (依頼書, S) and 依頼書正当 (依頼書, S) ;

依頼書正当 (依頼書, S) == 本数 (依頼書)¿0 and
       Disjoint'id(トップ (未出庫依頼書リスト (S)), 依頼書) ;
```

保存在庫ファイル (ファイルセット (S T_3)) == file @@**text** TL1;

保存依頼書ファイル (ファイルセット (S T_3)) == file @@**text** TL1;

ここで S は " 初期化 (T_1(irai) | T_2(tumini))*" なる系列をとる変数である.

irai, tumini, file は それぞれ 依頼書入力に対する処理の引数である依頼書, 積荷表入力に対する処理の引数である積荷表, ファイルの値を表す変数である[7]. 保存終了 T_3で終る処理系列に対しては保存すべき 在庫情報, 依頼書情報の内容を記述したファイルに興味があり, その他の系列については 出力である出庫指示書や, 出庫不可連絡を記載した 出力ファイルに興味がある. テキスト $TL1$ の (T_1, T_2, T_3に対する) 正しい実現を決めればすべての遷移列に対するそれらの値が定まる . T_1, T_2, T_3はそのような合同関係のうちいずれと一致するように実現してもよいということを表している.

これらの拡張射影文と初期化 (イ), 及び在庫管理全体の記述 (ハ), 計算指定 (ニ) で第 1 レベルの記述となる.

(ハ) では入力に応じて 3 つの状態遷移関数のいずれかを選択し, その後再び入力を得て繰り返す.

(ニ) においては, 入力系列に対して興味あるファイルの値に関する合同関係のみを指定する. 初期化, 計算指定, 在庫管理全体の記述の実際を表 8に表す.

3.5.5 第 2 レベルの記述例

第 2 レベルでは, 積荷表入力に対する処理を, **積荷表追加処理**と, **全未出庫依頼書に対する処理**を用いて詳細化した. これらの処理の要求性質を記述し, この順で状態遷移を行なうことにより, 積荷表入力に対する処理を展開する.

積荷表追加処理の要求記述は, 処理後のコンテナリストは積荷表追加処理の引数である積荷表の内容と処理前のコンテナリストの内容を合わせたものになっているということを記述する.

[7]変数に特定の表現式だけを代入するように制約を設けたい場合, 本文のように「irai は値だけが代入される変数である」というようにデータタイプ名で指定する. これだけでは対応できない場合例えば Sについては条件つき拡張射影を用いて valid(S) を満たすような S についてのみ拡張射影をとるというようにする. 厳密には本例についてもこの方法を適用すべきであるが簡単のため省略する.

表 8: 在庫管理の第 1 レベルの記述 (一部)

```
(* 初期化 *)
コンテナリスト (初期化 (fileset)) ==
                    ReadList(finit,'StockList.save', コンテナ LType) ;
未出庫依頼書リスト (初期化 (fileset)) ==
                    ReadList(finit,'UnprOrder.save', 依頼書 LType) ;
指示書          (初期化 (fileset)) == [ 0, snill, snill, 新リスト (指示 LType) ] ;
ファイルセット (初期化 (fileset)) == fileset ;
define 目的項 := 'ファイルセット (初期化 (finit) 処理 (input_seq ))' ;
(* 計算指定項 *)
CO1: 酒屋 (input_sq)== [
 Get_file( sakaya(input_sq), 出力ファイル ),
 Get_file( sakaya(input_sq), 'StockList.save' ),
 Get_file( sakaya(input_sq), 'OrderList.save' ) ] ;
CO2: sakaya(input_sq)==目的項 ;
define ' テール '  := ' Tail ( input_sq ) ' ;
(* 本処理 *)
P1: S 処理 ( input_sq )  ==
if      Get_cmnd( input_sq ) = 'quit'
then ( S 保存終了 )
else if Get_cmnd( input_sq ) = 'CargoTable'
then ( S 積荷表入力に対する処理 (Get_tumini(テール)) 処理 (Tail(テール)))
else if Get_cmnd( input_sq ) = 'ShippingOrder'
then ( S 依頼書入力に対する処理 (Get_irai(テール))   処理 (Tail(テール)))
else if not( Get_cmnd( input_sq ) = NULL )
then ( S 処理 (テール) ) ;
```

全未出庫依頼書に対する処理の要求性質では, **表 5**をこの状態遷移に適用した内容 (すなわち**表 6**とほぼ同内容) を記述する (**表 9**).

その他の記述は, テキスト $TL1$ を継承する. 以降のレベルの記述では, 処理の要求を記述する状態遷移関数と, 状態成分関数については, 第 1 レベルと同様に, 拡張射影文を適用することとする.

3.5.6 第 3 レベルの記述例—出庫処理を中心に—

第 3 レベルでは, まず, 状態成分関数として**指示書** (S), **在庫有り** (S,hon,hin) を追加した. 指示書 (S) は, 出庫指示書の内容を逐次求めていくために用いる状態成分関数である. 在庫有り (S,hon,hin) は, 出庫依頼書に記載されている本数 hon, 品名 hin に対して在庫があるかどうかを判定する述語関数である. 在庫がある場合, その依頼書に対応する (その商品をどのコンテナから何個ずつ出庫するか, また, その出庫によりそのコンテナが空になるかどうか等を記載した) 出庫指示書を作成し内部情報等を更新する **出庫処理**の要求性質 (ホ) を

表 9: 全未出庫依頼書に対する処理の要求記述

```
(* 未出庫依頼書リストに関する性質 *)
define 前提 := valid(S 全未出庫依頼書に対する処理) ;
define 処理後指示書リスト :=
  '差リスト (ファイルセット (S),
            ファイルセット (S 全未出庫依頼書に対する処理))';
define 処理後未出庫依頼書リスト :=
          '未出庫依頼書リスト (S 全未出庫依頼書に対する処理)' ;

I1: 前提 imply( Memberof(id, 処理後指示書リスト) ⊃
    Memberof(id, 未出庫依頼リスト (S)) and
  依頼書対応 ( 参照 (id, 処理後未出庫依頼リスト),
             参照 (id, 処理後指示書リスト))) == TRUE ;
I2: 前提 imply 集合 (処理後未出庫依頼リスト) =
    集合 (未出庫依頼リスト (S)) -
    依頼書集合 (処理後指示書リスト) == TRUE ;
I3: 前提 imply(Memberof(id, 処理後未出庫依頼リスト) ⊃
 not 出庫可能 ( 参照 (id, 処理後未出庫依頼リスト),
 コンテナリスト (S 全未出庫依頼書に対する処理)) == TRUE ;

 (* コンテナリストの性質 *)
define 処理後コンテナリスト :=
  'コンテナリスト (S 全未出庫依頼書に対する処理)' ;
define 指示書リスト:= 処理後指示書リスト ;
define ∃搬出指示 := ∃搬出指示コンテナ (指示書リスト, CargoID) ;
define '在庫減少量' :=
  '本数'在庫. 番号. 品名 (コンテナリスト (S), CargoID, hin) -
 仮想数量 ( hin, 表'在庫. 番号 (処理後コンテナリスト, CargoID) )' ;
C1a: 前提 imply
( ∃Memberof(CargoID , 指示書リスト) and not ∃搬出指示
 and Memberof(CargoID, コンテナリスト (S)) ⊃
 Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C1bi: 出庫可条件 imply
( ∃Memberof(CargoID, 指示書リスト) and not ★搬出指示 and
  and Memberof(CargoID, コンテナリスト (S))
  and ∃Memberof(hin, 指示書リスト) ⊃
  Uniq'id(処理後コンテナリスト) and
  在庫減少量 = 総本数'指示書. 品名. 番号
            (指示書リスト, hin, CargoID) ) == TRUE ;
C1bii: 前提 imply
( Memberof(CargoID, 指示書リスト) and not 搬出指示
 and Memberof(CargoID, コンテナリスト (S))
 and not ∃Memberof(hin, 指示書リスト) ⊃
 Uniq'id(処理後コンテナリスト) and
 コンテナ不変 (hin,
    表'在庫. 番号 (処理後コンテナリスト, CargoID),
    表'在庫. 番号 (コンテナリスト (S), CargoID)) )== TRUE;
C2a: 前提 imply
( not ∃Memberof(CargoID, 指示書リスト)
  and Memberof(CargoID, コンテナリスト (S)) ⊃
  Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C2b: 前提 imply
( not ∃Memberof(CargoID , 指示書リスト)
  and Memberof(CargoID, コンテナリスト (S)) ⊃
  Uniq'id(処理後コンテナリスト) and
  コンテナ不変 (hin,
     表'在庫. 番号 (処理後コンテナリスト, CargoID),
     表'在庫. 番号 (コンテナリスト (S), CargoID)) )== TRUE;
C3: 前提 imply
( Memberof(CargoID , 指示書リスト) and ∃搬出指示
 and Memberof(CargoID, コンテナリスト (S)) ⊃
 Uniq( トップ ( SYOKKOCONT ) ) and
 not Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;
C4: 前提 imply
( not Memberof(CargoID, コンテナリスト (S)) ⊃
  Uniq( トップ ( SYOKKOCONT ) ) and
  not Memberof(CargoID, 処理後コンテナリスト) ) == TRUE ;

(* 注
∃Memberof(CargoID , 指示書リスト)
指示書リスト中にコンテナ番号が CargoID のものを一つでも対象としている
ものがあれば真. それ以外は偽.
                :
∃搬出指示コンテナ (指示書リスト, CargoID)
指示書リスト中にコンテナ番号が CargoID のものを対象としているものがあり
それが搬出マークをつけているときのみ真. その他は偽.
                :
総本数'指示書. 品名. 番号 (指示書リスト, hin, CargoID)
指示書リスト中でコンテナ番号が CargoID で, 対象品名が hin であるものの本数
の和. ないときは 0 を返す.
*)

(* 個々の指示書の正当性 *)

SS6: 前提 imply 指示書リスト正当 (コンテナリスト (S),
          ボトム (処理後指示書リスト) CargoID) == TRUE ;
```

記述する. また出庫処理以外の簡単な処理 (作成された指示書を出力ファイルに書き出す**指示書出力処理**等) の定義 (ロ), と合わせて第 2 レベルの**全未出庫依頼書に対する処理**, **依頼書入力に対する処理**の詳細化を行なう (ハ).

ここでは, 出庫処理の要求性質について述べる. 出庫処理の要求性質を次の通り記述する. 指示書について**表 5**における (S1)~(S5) に加え, **表 5**の (S6a), (S6b) において, 仮想出庫量を 0 としたもの ((S6') とする) を記述する.

コンテナリストに関しては, **表 5**の (C1)~ (C4) の内容を記述する. なお, (C1), (C2) は処理後にコンテナが存在する場合で, (C3),(C4) は存在しない場合である. これらの記述をほぼそのまま公理の形で記述する[8].

記述の実際は, **表 7** において 依頼書入力に対する処理 を出庫処理と読み変え, 出庫不可条件 を前提にしている公理を取り除いたものとなる (公理のラベル O1~O6 は S1~S6 ' に読み変える).

状態遷移関数 **在庫有り**の満たすべき性質として, 「在庫有りが真であるのは, 状態 Sでの コンテナリストに, 依頼書の品名が依頼書の本数以上あるときかつそのときに限る」ことを記述しておく.

3.5.7 第 4 レベル以降の記述例

第 4 レベルでは, 状態成分関数 在庫有りをこのレベルで新たに導入した状態成分関数**在庫品名数量表** (S) を用いて実現する. 在庫品名数量表は, 品名から直ちに (状態 S における) 在庫数量が分かるテーブルである.

コンテナリストを用いて在庫有りを記述しているが, 品名数量表を参照するように 在庫有り を実現することにより, 処理の高速化が行なえる.

第 4 レベルは, 第 3 レベルの記述のうち**在庫有り**に関する性質の記述を取り除き, 在庫有りの実現, 新しい状態成分関数のための追加公理, 在庫品名数量表の要求性質等の記述を加えたものである.

[8] イオタでは内容が 0 のコンテナを加えても良いという実現が可能でありあまり好ましくない. またこれらの関係を間接的に述べているため, わかりにくくまた間違いが生じ易い.

新しい状態成分関数 在庫品名数量表の 出庫処理, 保存終了処理に対する要求性質として, 例えば出庫処理後の在庫品名数量表は指示書に記載されている分だけ減っていることなどを記述する.

第 5 レベルで 保存終了処理と 出庫処理を詳細化する.

出庫処理の詳細化では, コンテナリストの先頭 (古い在庫) から順に調べ, コンテナリストを辿るという方針を採用する. 処理の高速化のため, 必要量出庫すればその時点で処理を終了する様に記述している[9]. ループの脱出条件のためのパラメータ hon は出庫依頼本数に初期設定されており, 現時点の着目コンテナの該当品目の数量分を毎回の繰り返しで減少させる. このとき, 高速化のためコンテナリストを新たに作り直すことを避けている[10](**表 10**). 以上のように高速化の工夫をしているが, それにもかかわらずこの実現が, 出庫処理の要求性質を満たしていることの証明が行なえる.

3.5.8 証明例

ここでは, 上記の記述に対する証明例を 2 つ簡単に述べる. ここで挙げた証明はいずれもこのプログラムのキーとなる大事な部分に対する証明である.

第 5 レベルが第 4 レベルの正しい実現であることの証明　ここでは特に, 出庫処理の実現の (詳細化の) 正しさの証明について公理 S5 を中心にのべる. 第 4 レベルの出庫処理の要求性質公理 S1~S6' は全て,

$PreQ(S, \text{依頼書})\mathbf{imply}Q(S, \text{依頼書}, S \text{ 出庫処理 (依頼書)}) == \mathbf{TRUE}$

の形で表現される.

出庫処理はコンテナを一つづつ処理するコンテナ判定主処理を繰り返すことによって実現されているため, 証明はコンテナ判定主処理の適用に関する帰納法を用いる.

例えば, 公理 S5 に対しては, Q は

$Q(s',\ irai,\ s) \equiv$ 総本数 (指示書 (s)) $=$ 本数 ($irai$)

[9] イオタでは, ループの脱出条件をリストの最後かどうかで判定しており, 出庫すべき本数を出し終えた後では無駄な繰り返しを行なっている.

[10] イオタの記述では, リストを新たに作り直しており, リストの大域化による高速化は望めない

表 10: 出庫処理の実現

```
define '該当品数' :=  '参照_key_2( 参照_2( コンテナリスト (S) ), hin )' ;
define 'カーゴ内総数' := '総和_2( 参照_2( コンテナリスト (S) ) )' ;
(* 出庫処理の実現 *)
P11: S 出庫処理 ( 依頼書 ) ==
  S 出庫前処理(依頼書) 出庫主処理 ( 本数(依頼書), 品名(依頼書) ) ;
(* 出庫主処理 *)
P12: S 出庫主処理 ( hon , hin ) ==
  if      hon <= 0
  then    S
  else  ( S コンテナ判定主処理 ( hon, hin )
            出庫主処理 ( hon - 該当品数, hin ) ) ;
(* コンテナ判定主処理 *)
P13: S コンテナ判定主処理 ( hon , hin ) ==
  if      該当品数 = 0
  then  ( S 出庫不要コンテナ処理 )
  else if 該当品数 >= hon
  then  ( S 出庫コンテナ処理 ( hon , hin) )
  else  ( S 出庫コンテナ処理 ( 該当品数 , hin ) ) ;
(* 出庫コンテナ処理 *)
P14: S 出庫コンテナ処理 ( 出庫本数 , hin ) ==
  if      hon < カーゴ内総数
  then    ( S コンテナ変更処理 ( hin , 出庫本数 ) )
  else    ( S コンテナ削除処理 ( 出庫本数 ) ) ;
define 'CargoID'  :=' 参照_1(コンテナリスト (S))' ;
define 'CargoCONT':=' 参照_2(コンテナリスト (S))' ;
(* 出庫前処理 *)
BS1: コンテナリスト ( S 出庫前処理(依頼書) ) == トップ ( コンテナリスト ( S ) ) ;
BS2: 指示書          ( S 出庫前処理(依頼書) ) ==
 [ 番号(依頼書), 送り先(依頼書), 品名(依頼書), 新リスト (pr_4(指示書(S))) ];
(* コンテナ変更処理  *)
CC1: コンテナリスト ( S コンテナ変更処理(hin,hon) ) ==
  変更+_2( コンテナリスト (S), 更新-_2( CargoCONT, hin, hon ) );
CC2: 指示書          ( S コンテナ変更処理(hin,hon) ) ==
 [ 番号(指示書(S)), 送り先(指示書(S)), 品名(指示書(S)),
                後尾追加(pr_4(指示書(S)), [CargoID,hon,FALSE]) ];
CC3: 在庫品名数量表 ( S コンテナ変更処理(hin,hon) ) ==
 更新-_2( 在庫品名数量表(S), hin, hon ) ;
(* コンテナ削除処理 *)
CD1: コンテナリスト ( S コンテナ削除処理 ( hon ) ) ==
  削除+ ( コンテナリスト ( S ) ) ;
CD2: 指示書          ( S コンテナ削除処理 ( hon ) ) ==
[ 番号(指示書(S)), 送り先(指示書(S)), 品名(指示書(S)),
              後尾追加(pr_4(指示書(S)),[CargoID,hon,TRUE]) ];
CD3: 在庫品名数量表 ( S コンテナ削除処理 ( hon ) ) ==
 更新-_2( 在庫品名数量表(S), 品名(指示書(S)), hon ) ;
(* 出庫不要コンテナ処理 *)
NC1: コンテナリスト ( S 出庫不要コンテナ処理 ) == 前進(コンテナリスト (S)) ;
(* 変化しない状態成分の公理は省略している *)
```

である. ここで "総本数" は指示書中の本数の総計を表す関数である. また不変式 R, E として 次の式を選んだ.

$$
\begin{aligned}
R(\ s',\ irai,\ \langle s,\ hon,\ hin\rangle) \equiv\ & \\
& [\text{総本数}(\ \text{指示書}\ (s)\ ) = \textbf{if} \quad hon > 0 \\
& \quad \textbf{then}\ \ \text{本数}\ (irai) - hon \\
& \quad \textbf{else}\ \ \text{本数}\ (irai)] \qquad (1) \\
& \wedge\ [\text{品名}\ (\ irai\ )\ =\ hin] \qquad (2) \\
& \wedge\ [hon > 0\ \supset\ hon\ \leq \text{本数\_在庫.品名}\ (\ \text{コンテナリスト}\ (s),\ hin\ )\ ] \qquad (3) \\
& \wedge\ [\textbf{length}(\ \text{コンテナリスト}\ (s)\ )\ \geq\ 0] \qquad (4)
\end{aligned}
$$

$E(\langle s, hon, hin\rangle) \equiv$ **length**(コンテナリスト (s))

length(コンテナリスト) はコンテナリストの現状ポインタ以降のリストの長さを与える基本関数とする.

式 (1) は作成中の指示書の総本数と引数の hon の間の関係を表したものであり, (2) と共に, 最終段階の証明で使用する. 式 (3) は次々と更新されていくコンテナリストと引数の hon の間の関係を表したものであり, (4) を帰納的に保証するために用いる.

なお, 証明には中間補題を設けたり, また証明する項が幾つかの項の論理積で表される場合は, 個々の項に分けて証明することもある. 例えば帰納段階の証明において (1),(3) の各々の証明に用いた中間補題は, 関数 $r(...)$ の適用により, 関数 総本数, 本数_在庫.品名の値がどの様に変化するか, ということを記述した

$$
\begin{aligned}
& \text{総本数}\ (\ \text{指示書}\ (\ r(\langle\ S,\ HON,\ HIN\ \rangle)\ )\ ) == \qquad (5) \\
& \qquad \textbf{if} \quad N(\ S,\ HIN\ )\ \geq\ HON \\
& \qquad \textbf{then}\ \ \text{総本数}\ (\ \text{指示書}\ (S)\ ) + HON \\
& \qquad \textbf{else}\ \ \text{総本数}\ (\ \text{指示書}\ (S)\ ) + N(\ \text{コンテナリスト}\ (S),\ HIN\ ) \\
& \\
& \text{本数\_在庫.品名}\ (\text{コンテナリスト}\ (\ r(\langle S,\ HON,\ HIN\rangle)\ ),\ HIN) \\
& \qquad == \quad \text{本数\_在庫.品名}\ (\text{コンテナリスト}\ (S),\ HIN) \\
& \qquad - \quad N(\text{コンテナリスト}\ (S),\ HIN) \qquad (6)
\end{aligned}
$$

である.

到達正当性条件から導けるであろう具体的な前提条件として以下の条件を用い, これが, 出庫処理後も満たされていることも確認した.

$R(\ s',\ irai,\ \langle s,\ hon,\ hin\rangle) \equiv$

$\forall id, \widehat{hin}[$**Unique_id**(トップ (コンテナリスト (s)))

$\wedge$ **Unique_id**(トップ (未出庫依頼リスト (s)))

$\wedge$ (**Memberof**$(\widehat{hin}, id,$ コンテナリスト $(s)) \supset$

本数_在庫.番号.品名 (トップ (コンテナリスト $(s)), id, \widehat{hin}) \geq 0$)

$\wedge$ (**Memberof**$(id,$ コンテナリスト $(s)) \supset$

Unique_hin(表_在庫.番号 (トップ (コンテナリスト $(s)), id)))$]

第 3 レベルが第 2 レベルの正しい実現であることの証明

第 2 レベルでの要求性質はいくつか存在するが, ここでは, 全未出庫依頼書に対する処理を行なった後の未出庫依頼書リストの中には出庫できるものは残されていないこと (公理 I3) の証明と, 全未出庫依頼書に対する処理を行なっても具体的な前提条件が保存されることの証明を行なった.

公理 I3 の証明

後置条件, 不変式 Q, Rは, 以下のように定め, 証明を行なった.

$Q(s)$ $\equiv$ Memberof$(id,$未出庫依頼リスト $(s))$ $\supset$

¬在庫有り $(s, pr_3($参照 (未出庫依頼リスト $(s))), pr_4($参照 (未出庫依頼リスト $(s)))$

$R(s', s)$ $\equiv$ Memberof$(id,$未出庫依頼リスト $(s))$ $\supset$

¬在庫有り $(s, pr_3($参照 (未出庫依頼リスト $(s))), pr_4($参照 (未出庫依頼リスト $(s)))$

中間補題として,

$$\begin{aligned}&\text{Memberof}(id, \text{コンテナリスト}\ (h(s, irai))) \supset \\ &\qquad \text{本数\_在庫.番号}\ (id, hin, \text{コンテナリスト}\ (h(s, irai))) \\ &\qquad \leq \text{本数\_在庫.番号}\ (id, hin, \text{コンテナリスト}\ (s)) \end{aligned} \tag{7}$$

を用いた.

上述の手順中にある値に関する議論も当然行なうべきであるが, 省略する. なお, 停止の証明に用いる $E(s)$ は, 出庫処理のときと同様に, 未出庫依頼書リストの現状ポインタから後尾までの長さを値とする関数を用いた.

到達正当性条件から導けるであろう具体的な前提条件として以下の条件を用い, これが, 全未出庫依頼書に対する処理を行なった後でも満たされていることも確認した.

証明には, すでに証明した「出庫処理後も前述の具体的な前提条件が保存されること」を補題に用いることにするが, このレベルでの具体的な前提条件として以下の条件を考え, このうちで, 「出庫処理」に関して証明されていないもの (8, 9) は, 新たに証明を行なった.

$$\begin{aligned}&\text{Uniq}(\text{未出庫依頼リスト}\ (s)) \\ &\wedge\ \text{Memberof}(id, \text{未出庫依頼リスト}\ (s)) \supset \\ &\qquad \text{本数\_依頼.番号}\ (id, \text{未出庫依頼リスト}\ (s)) > 0 \\ &\wedge\ \text{Uniq}(\text{コンテナリスト}\ (s)) \\ &\wedge\ \text{Memberof}(id, \text{コンテナリスト}\ (s)) \supset \\ &\qquad \text{Uniq\_hin}(\text{表\_在庫.番号}\ (id, \text{未出庫依頼リスト}\ (s))) \\ &\wedge\ \text{Memberof}(id, hin, \text{コンテナリスト}\ (s)) \supset \\ &\qquad \text{表\_在庫.番号.品名}\ (id, hin, \text{未出庫依頼リスト}\ (s)) \geq 0 \\ &\wedge\ \text{Memberof}(id, \text{コンテナリスト}\ (s)) \supset \\ &\qquad \text{表\_在庫.番号}\ (id, \text{未出庫依頼リスト}\ (s)) > 0 \end{aligned} \tag{8, 9}$$

表 11: 公理 I3 具体的前提条件の保存の証明に用いた検証支援系コマンド操作回数

	項書換え	公理入力	場合わけ	恒真性判定	その他	合計
公理 I3	6	18	24	15	14	77
具体的前提	12	18	18	18	15	81
条件の保存	12	9	13	11	11	56

証明作業量及び, 証明に関する考察

全未出庫依頼書に対する処理の性質のうち公理 I3 と, このレベルの具体的な前提条件の保存にの証明に要したコマンド操作回数に関するデータを**表 11**に示す.

表 11の I3 の行 (1 行目) は公理 I3 の証明に要した検証支援系のコマンド操作回数である.

表 11の 2 行目はこのレベルの証明のうち, 全未出庫依頼書に対する処理で具体的前提条件が保存することの証明に要したコマンド操作回数である. 補題として「出庫処理」において 具体的前提条件が保存することの証明で得られた結果と式 (8),(9) を用いている.

3 行目は (8),(9) の証明に要したコマンド操作回数である.

各証明に要した基本関数の補題の総数は, 公理 I3, 具体的前提条件保存それぞれに対し, 12, 20 個であった.

出庫処理の性質のうち指示書に関する性質 (表 5の公理 S1~S6 ') を証明したときの, 検証支援系のコマンド操作回数に関するデータを**表 12**に表す.

表 12は, 公理 S5 の証明に用いた検証支援系コマンド操作回数をまとめたものである. 1 行目の数値は公理 S5 の証明に要した回数である. 2 行目は, 停止性の証明に用いた検証支援系コマンド操作回数をまとめたものである.

表 13の 1 行目は公理 S3 の証明に要したコマンド操作回数であり, 以下も同様である. 公理 S3, S4, S6 ' , 具体的前提条件の保存の証明では中間補題をそれぞれ, 1, 3, 3, 6 個使用した. 公理 S1, S2 は S3 とほぼ同様の回数と思われる. また公理 S3 等の証明で関数が値をもつか, 再帰が停止するかなどの議論は公理 S5 のそれと同様なので, 公理 S5 以外は部分正当

表 12: 公理 S5 の証明に用いた検証支援系コマンド操作回数

	項書換え	公理入力	場合わけ	恒真性判定	その他	合計
公理 S5	36	53	32	32	38	191
停止性	16	6	21	9	10	62

表 13: 公理 S3, S4, S6', 具体前提条件の保存の証明に用いた検証支援系コマンド操作回数

	項書換え	公理入力	場合わけ	恒真性判定	その他	合計
公理 S3	8	10	8	0	11	37
公理 S4	36	14	30	22	16	118
公理 S6'	54	78	71	47	49	299
具体前提条件の保存	88	46	105	46	38	323

性の証明のみ行なった. 明らかに同一の証明になる項の証明も省いている. 具体的前提条件保存の証明で, 中間補題の証明に要したコマンド数が多い原因は, 証明の難しさではなく, 中間補題の数の多さによるものである.

各証明に要した基本関数の補題の総数は, 公理 S3, S4, S5, S6, 内部前提条件保存それぞれに対し, 4, 12, 21, 27, 15 個であった. またそれらの補題を重複させずに, 合計した総数は, 66 個である. 一般に, 抽象度の高い基本関数を採用すると, 基本関数の補題が多く必要となるが, 証明自身は短くなる. 本論文のプログラムではポインタ込みのリストを基本関数として使用しており, リストの実現の議論は (本論文と直接関係ないので) していない. (もし議論するとしても, リストの実現だけを別に扱う方がよい.)

CPU 時間

証明に要した CPU 時間を Sun Sparc IPC を使用して計測した値を**表 14**に記す.

表 14: 検証支援系 CPU 時間 (秒)

S3	S4	S5	S6'	停止性	level4 での保存	I3	level2 での 保存
30	330	540	1100	150	910	270	370

公理 S6 の証明に CPU 時間がかかっているのは, 主として項の大きさに原因がある. 中間補題の数は 3 つであるが, 実質は 7 つぐらいの項の内容と大きさを持っているとみなせる. 項の大きさは, 恒真性判定や, 項の書き換えなどの実行時間に影響を及ぼす.

3.5.9 考察

実用性を考慮に入れてプログラム設計開発方法を提案する場合, 言語, 設計方法, 詳細化の正しさ等の定義の明確さ, 記述の自然さ, 例題の適用結果, 処理系の実行効率, プログラムの正しさの検証法 (証明が計算機の支援のもとで行えること) 等が重要である. 他手法との比較を, 特に設計法, 適用結果, プログラムの実行効率に焦点をしぼって行なう.

設計法に関しては, トップダウンによる階層的設計を提案しているものは多い. しかしながら, そのレベルで意味定義がきちんと閉じている階層化を行なっているもの (例えばイオタ, HIPS[13]) はわずかであった.

実行環境システムに関しては, 言及されているもの (例えば, Stella, イオタ, Valid[13]) はいくつかあった. これらのうち関数型言語の処理系はインタプリタであり, 実行効率はあまり良くないと予想される.

プログラムの検証も行なっているが[36] プログラムの検証可能性に関しては処理系がサポートされているのはイオタだけであった. しかし, いくつかの脚注でふれたように, イオタでも我々の観点からでは好ましくない点があり, プログラムの正しさの証明も実例についてはふれられていなかった. また, 我々の第一レベルに相当する本来の要求記述を行なった例も見られなかった.

3.6 結言

この章では, ASM の階層的設計における有用な概念である拡張射影と到達正当性条件を提案し, その手法を用いて, 在庫管理の記述を自然に行なうことができた. また, これらの概念の実用上有効な証明方法を提案した. なお, 最終的に得られた例題の実行速度は C 言語で普通に記述したものと比べ, 遜色のないことを確認している.

本論文では完全に与えられた要求仕様からの詳細化について述べた. このように完全な要求仕様が与えられたもとでの詳細化を研究の対象にしているのは, そのような場合においても現実的に「形式的な詳細化」が可能であることを示すを一つの目的としているからである.

ここで扱っている形式的な階層的設計法は不完全な要求仕様からは詳細化できないということはない. 本手法でも, 例えば, 状態遷移名のみ導入し, その状態遷移に関する細かな要求を記述せずに, 以降詳細化するという詳細化も可能である. 特に重要な部分にのみ要求性質を記述し, 詳細化を議論するより現実的な詳細化法も考え得る. 一般には後者の設計法は詳細化のたびに, 設計者の決定が入っていくだけであり, 本論文で提案している意味での「詳細化の正しさ」を証明するのは簡単であると考えられる.

また, 現実問題としては, プログラムの要求仕様の変更への対処法をどのようにするかという問題も考え得る. プログラムの要求仕様の変更の際は, 一般にはその変更のあったレベルから詳細化を再度行なうこととなる. このときは, 各状態遷移の前提条件が具体的に定まっている場合は各状態遷移関数は独立なものとして捉えることができ, 新しい要求仕様への変更に必要な状態遷移を独立に, 新しいものに置き換え, その置き換え の妥当性を形式的に証明するという方法が考えられる.

4 信頼できる通信環境における分散実行動作仕様群の自動導出

4.1 序言

本章では, 分散実行動作仕様群の自動導出法について述べる. 本章で提案するモデルでは, 非決定性の動作を許すように拡張する. また, 提案する自動生成アルゴリズムでは, 与えられた全体仕様と設計者が指定したリソース (レジスタや入出力ゲート) の複数分散配置から, 各ノードの動作仕様の導出を行う. 各ノードの動作仕様は, 全体仕様の各状態遷移を一定の方針にしたがってノード間のメッセージの送受信動作などを行なう幾つかの状態遷移の系列に変換することによって導出する. その際, 各状態遷移におけるノード間の送受信動作の総数が最小になるように各ノードの動作仕様を導出しており, 効率よく動作仕様を実行できる.

4.2 諸定義

ここで全体仕様, 実現環境, 等価性, 導出問題等いくつかの概念を述べる.

4.2.1 全体仕様

以降の章では, 入出力動作は各状態遷移に付随し, ゲートに対して行なわれるとし, 非決定的選択が可能であるとする. 入出力動作は次の 3 種類からなる.

(1). 入力動作 $a?x$ はゲート a からのデータ入力を表す. 入力値は, 入力変数 x に代入される. x のスコープはその状態遷移の範囲とする.

(2). 出力動作 $a!E(\ldots)$ で式 $E(\ldots)$ の値をゲート a に出力することを表す. Eは Rの要素である (複数の) レジスタを引数として持つ関数である.

(3). i で外部から観測されない動作を表す.

このモデルでは各状態遷移関数は $s - \langle a?x, C, CR\rangle \rightarrow s'$で表す. ここで $a?x(a!E(\ldots)$ も可) は動作であり, Cはのレジスタ $R_{k_1}, \ldots, R_{k_t}$と入力変数 x を引数とする述語であり, CRは次の状態 s'でのレジスタ値を定める代入文 $R_i \leftarrow f(R_{k_1}, \ldots, R_{k_t}, x)$ の組である. Cを遷移

条件と呼び, CRをレジスタ更新式と呼ぶ. init は初期状態 s_1と各レジスタの初期レジスタ値を指定する.

初期状態と初期レジスタ値から以下のように動作を行なう. いま現在の状態を s_i, レジスタ値を $v_{R_1},\ldots,v_{R_n}$とする. 状態遷移関数 $s_i - \langle a?x, C, CR\rangle \rightarrow s_j$において, もし現在のレジスタ値 $v_{R_1},\ldots,v_{R_n}$および, 入力変数値 v_xが遷移条件 Cを満たせば $a?x$ が実行可能となる. s_iからのすべての状態遷移関数について実行可能な動作を求め, 実行する動作を一つ非決定的に選択する. $a?x$ の実行後, 状態 s_jに遷移し, レジスタ更新式にしたがって各レジスタ値を更新する (出力動作の場合, 出力後に更新を行なう). 例えば, $R_1 \leftarrow R_1 + R_2 - x \in CR$の場合, レジスタ R_1の新しい値は, $v_{R_1} + v_{R_2} - v_x$になる. このモデルでは, 入力データの内容に依存して実際に入力動作を行なうかどうかを決定することができる.

[例 3]

図 6に 全体仕様 SSの例 $S1$ をあげる. ここではグラフ表現している. 各ノードと枝はそれぞれ状態と状態遷移を表す. 状態遷移を表す枝には, 動作, 遷移条件, レジスタ更新式の 3 字組がラベルとして付けられている. レジスタは, $R_1,\ldots,R_6$の 6 つである. また, 使用されているゲートは, a, b, c の 3 つである.

この例は, 店のレジの業務をモデル化したものである. 状態 S_1が初期状態であり, 各レジスタは $[0, 0, \mathrm{data}, 0, 0, 0]$ に初期設定されている (data は商品データベースの内容を表す). 商品番号読みとり装置 (ゲート a) から商品番号を読み, レジスタ R_1に保存する ($S_1 \rightarrow S_2$). 状態 S_2で商品番号 (R_1の値) が 0 でなければ, 商品データベース R_3から該当商品番号の価格 $R_3(R_1)$ を調べレジスタ R_6に代入する ($S_2 \rightarrow S_3$). この状態遷移では出力はないので動作は $b(b!\emptyset$ のこと) としている. 表示器 (ゲート b) に商品番号, 価格 (レジスタ R_1, R_6の値) を表示し, 総合計 R_2を更新し, 初期状態に戻る ($S_3 \rightarrow S_1$). 一方状態 S_2でレジスタ R_1の値が 0 であれば商品入力の終了と見なし, 総合計 R_2を表示器に表示し ($S_3 \rightarrow S_4$), 客から受けとった現金を順次レジ (ゲート c) に入力していく ($S_4 \rightarrow S_4$). 金額の総合計はレジスタ R_4に入る. 金額の総合計 R_4が商品価格の総合計 R_2以上になれば, 釣り銭をレジに表示し, 店の売上の総計 R_5を計算し, 各レジスタをリセットして初期状態に戻る ($S_4 \rightarrow S_1$). □

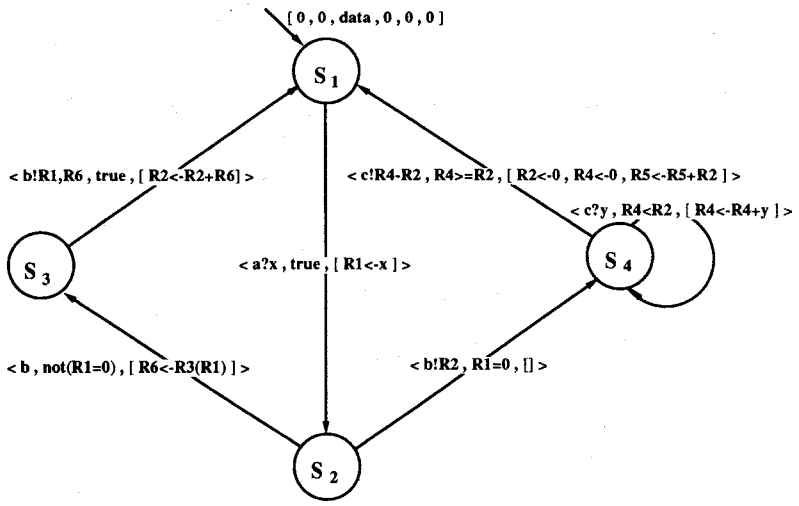


図 6: 全体仕様の例 $S1$

Fig.6 Sevice Specification $S1$

4.2.2 分散実行環境

与えられた全体仕様を p 個のノードからなる分散システム上で協調して動作するシステムとして実現する (図 7).

本論文では, 各ノードの動作仕様も上述の順序機械型モデルとしてモデル化する. 各ノード kの動作仕様を PE_kで表す. p 個のノードの動作仕様の組を PE^{1-p}で表し (p ノードの) 分散システムの動作仕様群と呼ぶ (プロトコル仕様とも呼ばれる[21]).

m 個のレジスタ及び入出力ゲートがそれぞれどのノードに属するかをユーザが指定し, それらの割当をσで表す. 各入出力ゲート, 各レジスタは必ずいずれかのノードに属するとする. 同一ゲートが複数のノードに属することはないと仮定するが, 同一レジスタが複数ノードに属してももよい (図 7のレジスタ R_2). これはリソースの複数分散配置に相当する.

PE_iから PE_jへの通信路は無限の容量を持つ FIFO キュー (queue_{ij}) で結ばれているとし, 両端のゲート名を g_{ij}とする. よって, PE_iが "g_{ij}!data" を実行すると queue_{ij}に data の値が入る. また queue_{ij}に要素があるときに PE_jが "$g_{ij}?x$" を実行すると, queue_{ij}の先頭にある要素が変数 "x" に代入され, その要素が queue_{ij}から削除される.

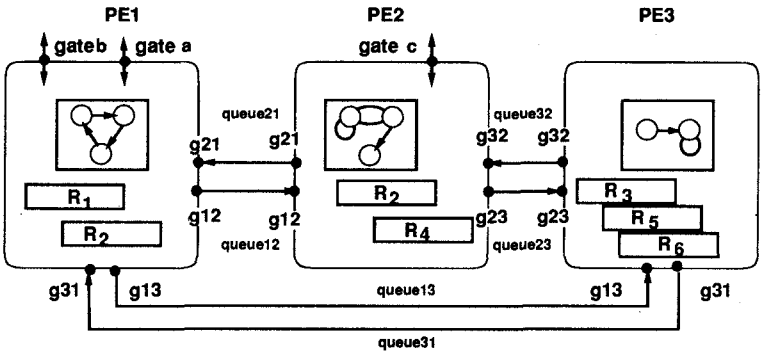


図 7: 分散環境モデル

Fig.7 Distributed System

4.2.3 動作仕様群導出問題

分散システムの動作仕様群 PE^{1-p}において，各 $PE_k(1 \leq k \leq p)$ が初期状態でかつ，通信路の各 FIFO キューがすべて空であるときを，PE^{1-p}の初期状態とする．分散システムの全体仕様 SSと，分散システムの動作仕様群 PE^{1-p}が等価であることを以下の様に定義する．

[定義 10] [等価性]

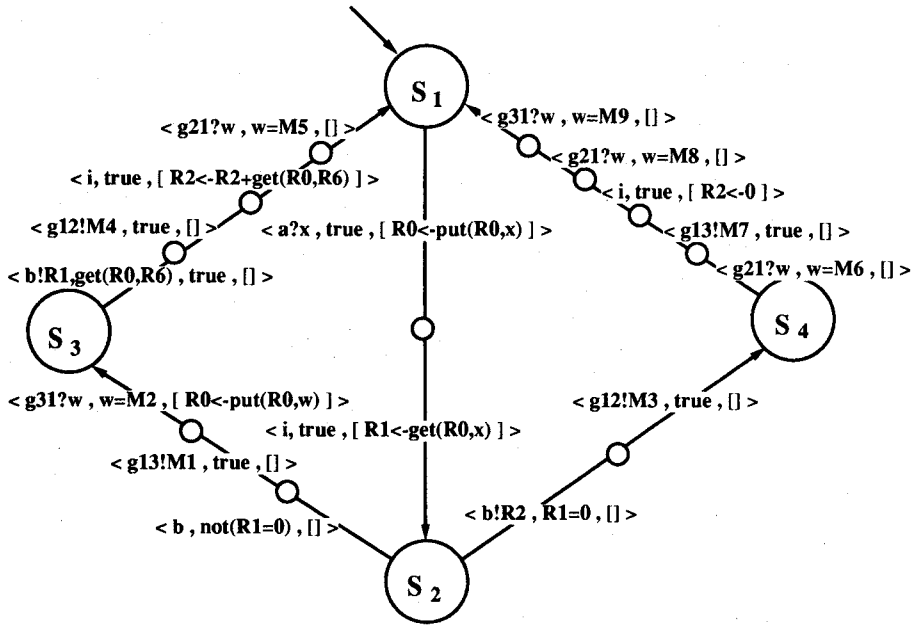
分散システムの動作仕様 PE^{1-p}において，各 PE_i, PE_j間の通信に用いられる送受信動作 $g_{ij}?x, g_{ij}!E(\ldots)$ を，観測不可能な動作とし，その他の動作を観測可能な動作とする．このとき 全体仕様 SSと PE^{1-p}が観測合同[29, 30]であれば，両者は等価であるという． □

二つの状態機械 M_1, M_2が観測合同であるというのは，M_1で実行できる任意の観測可能な動作の系列が M_2でも実行でき，それが M_1, M_2を入れ替えても成り立ち，かつ任意の実行可能な観測可能動作系列を行なった時点での実行可能かつ観測可能な動作が互いに等しいことである．

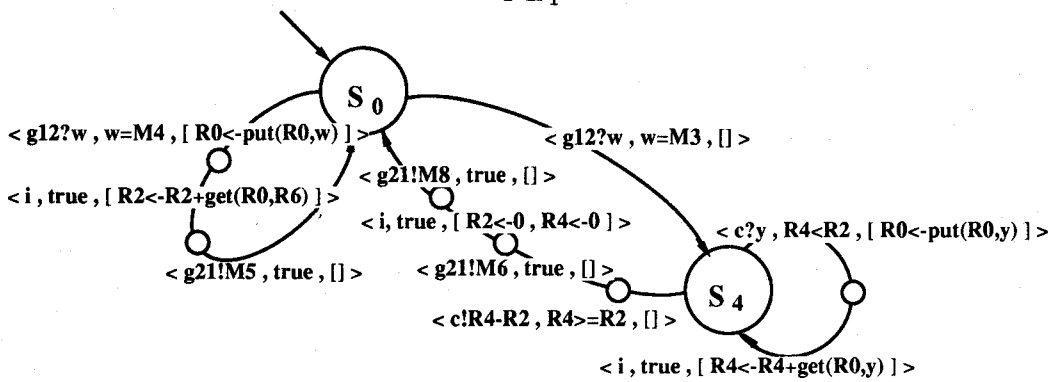
図 6の例で，ノードを 1,2,3 とし，各ゲートとレジスタの割当σを次のように決める．

	ノード 1	ノード 2	ノード 3
レジスタ	R_1, R_2	R_2, R_4	R_3, R_5, R_6
ゲート	a, b	c	

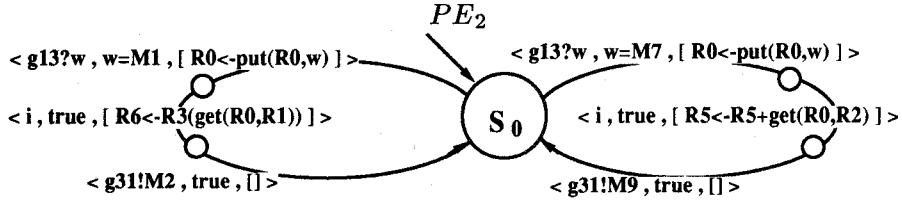
[例 4]



PE_1



PE_2



PE_3

$$M_1 \equiv \mu^2, \{\langle R_1, v_1\rangle\} \quad M_2 \equiv \xi^2, \{\langle R_6, v_6\rangle\} \quad M_3 \equiv \xi^4$$

$$M_4 \equiv \mu^3, \{\langle R_6, v_6\rangle\} \quad M_5 \equiv \xi^3 \quad M_6 \equiv \mu^5$$

$$M_7 \equiv \beta^5, \{\langle R_2, v_2\rangle\} \quad M_8 \equiv \xi^5 \quad M_9 \equiv \xi^5$$

図 8: 各動作仕様 PE_k

Fig.8 Protocol Entities' Specifications

この割り当てσに対して，図 8で表す 3 字組 PE^{1-3}は図 6の $S1$ と等価である．

例えば，図 6の $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1$の状態遷移系列に相当する各 PE_kの動作は次の通りである．但し，各 PE_kには σで割り当てられたレジスタに加え作業用のレジスタ R_0があり，他ノードから送られたメッセージを一時的に格納しておくために利用する．

PE_1は状態 S_1から始め，ゲート a から x を読み込みレジスタ R_1に代入し，S_2に遷移する ($\mathrm{put}(R_0, m)$ は入力 m をレジスタ R_0に格納する補助関数．$\mathrm{get}(R_0, R_h)$ はレジスタ R_0からレジスタ R_hの最後に格納された値を取り出す補助関数．入力変数 x の値を取り出す 関数 $\mathrm{get}(R_0, x)$ も同様に定義する)．ここで，動作 g_{11}は， レジスタ更新 ($R_1 \leftarrow \mathrm{get}(R_0, x)$) のみを行うための非観測動作を表すために便宜上，このような記述を行っているものである．実際にゲート g_{ii}があるわけではない (以下，同様)．この間 PE_2, PE_3は状態 S_0のまま．次に，PE_1は状態 S_2で $R_1 \neq 0$ の判定を行ない，PE_3に R_1の値をメッセージ M_1で送る．これを受けとった PE_3は R_6の値を計算し，その値をメッセージ M_2として PE_1に送り S_0にもどる．PE_1は M_2を受けとった後，状態 S_3に入り，ゲート b から，R_1と今受けとった R_6の値を出力し，PE_2に R_6の値をメッセージ M_4として転送する．PE_2は M_4を受けとり，これを用いてレジスタ R_2を更新し，更新終了メッセージ M_5を PE_1に送信し，S_0にもどる．PE_1は M_5を受けとったのち S_0にもどる．各メッセージの内容については **4.3.2節**で述べる． □

[定義 11] [導出問題]

全体仕様 SSと分散システムのノードの集合 {ノード$_1$,..., ノード$_p$} 及び，ゲートやレジスタの割当σが与えられたとき，SSと等価な p ノードの分散システムの動作仕様 PE^{1-p} を導出する問題． □

但し，全体仕様として与えられる SS及び，σは次のような 2 つの制約条件を満足するものとする．

[条件 2]

(1). 与えられた SSの各状態で 2 つの動作 "a..." と "b..." (a, b はゲート名) が記述されていたら，ゲート "a" と "b" の属するノードは同一でなければならない．

(2). 初期状態を s_Iとする. $s_I - \langle a \cdots, C(\ldots), CR\rangle \rightarrow s'$ なるすべての状態遷移に対し，遷移条件 $C(\ldots)$ は，ゲート "a" が属するノードに属するレジスタ $R_{v_1}, \ldots, R_{v_t}$のみを用いた述語で記述されていなければならない．また，初期状態からの出力動作で必要なレジスタもゲート "a" が属するノードに属していなければならない．

(3). 全体仕様 SSは，内部動作を含まない □

条件 2の (1) を満足しないと，ある状態で実行可能な動作を実行するノードが複数個存在することになり，その場合，それらの中の一つの動作を実行するには，ノード間で合意をとる必要がある．この合意は例えば，ノード間で一つのトークンを回して，そのトークンを獲得したノードが動作を実行できるようにすれば解決するが[24]，以下での議論を簡単にするため，ここでは上記の条件 **2**の (1) を設ける．

条件 2の (2) は SSが最初に動作を始めるときにその動作の実行可能性判定，及びゲートへの出力を自ノードで行なえることを表しており，本質的な制約ではない．**条件 2**の (2) を満足しない場合，特別な状態を導入しその状態を初期状態とみなし，ダミーの動作 (その動作の実行は何時でも可能とする) でもとの初期状態に遷移するように SSを変形すれば，**条件 2**の (2) を満足する．

条件 2の (3) は，以降の議論の簡単化のため設ける．本質的な制約ではない．

条件 2の (1) より，各状態 s_iで実行可能な動作は 1 つのノードで実行される．このノードの動作仕様を Snode(s_i) で表し，(状態 s_iの) 責任ノードと呼ぶ．

また各レジスタ R_hの属するノードの PEの集合を Rnode(R_h) で表す (一般に R_hの属するノードは複数考えられるので集合としている)．さらに，状態 s_iから始まる状態遷移の集合について，(イ) 動作の実行可能性を判定する遷移条件の引数に指定されたレジスタ名，(ロ) 出力動作の引数に用いられるレジスタ名，及び (ハ)Snode(s_i) において (σで指定されて)，保持すべきレジスタ名の集合和を Cset(s_i) で表す．

Snode　Snode(S_1) $\equiv PE_1$, Snode(S_2) $\equiv PE_1$, Snode(S_3) $\equiv PE_1$, Snode(S_4) $\equiv PE_2$

Rnode　Rnode(R_1) $\equiv \{PE_1\}$, Rnode(R_2) $\equiv \{PE_1, PE_2\}$, Rnode(R_3) $\equiv \{PE_3\}$,
Rnode(R_4) $\equiv \{PE_2\}$, Rnode(R_5) $\equiv \{PE_3\}$, Rnode(R_6) $\equiv \{PE_3\}$

Cset　Cset(S_1) $\equiv \{R_1, R_2\}$, Cset(S_2) $\equiv \{R_1, R_2\}$, Cset(S_3) $\equiv \{R_1, R_2, R_6\}$, Cset(S_4) $\equiv \{R_2, R_4\}$

となる.

4.3 動作仕様群の導出

4.3.1 基本方針

まず, 導出する各ノードの動作仕様 PE_k の動作の概略について述べる.

後で状態遷移図の縮約を行なうが, 基本的に 各 PE_k はもとの SS と同じような形の状態遷移図を持ち, もとの SS の一つの状態遷移を通信のための幾つかの状態遷移の系列に置き換えることにより構成する. また, レジスタ更新の方法は色々考えられるが, 更新すべきレジスタを持つ各ノードが, 計算に必要な引数のレジスタ値を他ノードから受け取り, 自ノードで計算後更新を行なうことにする. 同一レジスタを複数のノードが個々に保持している場合でも, 一つのノードでのみ計算して計算結果を他ノードに通信するということはせずに各ノードで更新を独立に行なう. このとき, 各ノードは SS の一つの状態 s_i に対して, 次のことを行なえば良い.

(イ) s_i の責任ノード Snode(s_i) が実行可能な状態遷移 $s_i - \langle A, C, CR\rangle \rightarrow s_j$ を一つ選択し, 動作 A を実行すること.

(ロ) 選択された状態遷移のレジスタ更新式 CR に従って各ノードのレジスタ値を更新すること.

(ハ) 状態 s_j の責任ノード Snode(s_j) が,

(a) 次に自分が (責任ノードとして) 状態 s_j のことを実行すべきことを知ること,

(b)(他ノードでの) すべてのレジスタ更新が終了し, 状態 s_j に相当する状態に到達したことが分かること, 及び,

(c) 状態 s_j での次の動作を決定するのに必要なレジスタ値の情報 (Cset(s_j)) を知ること.

条件 2 の (1) より, (イ) の選択は Snode(s_i) 自身で行える. 但し, 判定に必要なレジスタ値はすべて Snode(s_i) が知っていることを仮定しておく (**条件 2** の (2) より初期状態ではこの仮定は成り立つ).

上記 (ロ) のためには，更新すべきノードは，(i) どの更新式に従って更新すべきか，という情報が必要である．またレジスタ値を送信すべきノードは，(ii) どこへどのレジスタ値を送るべきかという情報が必要である．各ノードがこれらの情報を (直接，あるいは間接的に) 責任ノードからのメッセージとして受けとるために，以下で述べるメッセージすべてに状態遷移を識別するラベルを付加して送信することにする．これにより各ノードがメッセージを受信した際に，上記 (i)，(ii) の情報を得ることが可能となる．

あとで，通信のコスト基準に合わせて，メッセージの削減を行なうが，その評価基準としては，各状態遷移ごとに得られた状態遷移の系列で交換されるメッセージの総数のみを考える．従って，メッセージの内容 (メッセージの長さ) は評価基準にはいれない．この仮定は実際のシステムを考えても妥当と思われる．

いま，Snode(s_i) が状態遷移 $s_i - \langle A, C, CR\rangle \rightarrow s_j$ を選択したとする．各 PE_k は，(イ)，(ロ) を次の 5 つのフェーズで実現することにする．(1) Snode(s_i) が動作 A を実行する．(2) Snode(s_i) からレジスタ値を送るノードへきっかけを与えるメッセージ (以後 α型メッセージと呼ぶ) を送る．(3) α型メッセージを受けとったノードからレジスタ値を必要とするノードへレジスタ値の情報を持ったメッセージ (以後 β型メッセージと呼ぶ) を送る．(4) 各ノードがレジスタ更新式に従ってレジスタ値を更新する．(5) レジスタ更新を行なった各ノードから更新が終了したことを知らせるメッセージ (以後 γ型メッセージと呼ぶ) を Snode(s_j) に送る．

但し，レジスタ更新を自ノードのレジスタのみを用いて行なえるノードにはβ型メッセージ等が届かないので，フェーズ (2) で更新のきっかけを与えるメッセージ (λ 型メッセージ) を Snode(s_i) から送ることにする．さらに，Snode(s_i) が知っている (Cset(s_i) に含まれる) レジスタの更新前の値や入力変数の値をそれを必要とするノードにフェーズ (2) において送ることも考えられる．このメッセージをη型メッセージとする．

上述の (ハ) の (a),(b) については 上述のフェーズ (5) で，Snode(s_j) がレジスタ値の更新を行ったすべてのノードからγ型メッセージ (状態遷移の識別ラベル付) を受け取ることにより，自ノードが次の責任ノードであり，かつ，現状態遷移が終了し s_j に対応する状態に遷移したことを知ることが出来る．但し，次責任ノード Snode(s_j) へまったくγ型メッセージが

送られず，かつ，以下で説明するρ型メッセージも送られない場合は，現責任ノードが次責任ノード Snode(s_j) へ直接メッセージを送ることにする．このメッセージをθ型メッセージと呼ぶことにする．

(c) については，Snode(s_j) が Cset(s_j) に含まれるレジスタのうち自ノードが保持していないレジスタの値を知る必要がある．そのレジスタ値 (例えば，v_{R_h}) が今の遷移で更新されたものなら，更新後の値を上述のフェーズ (5) で，Rnode(R_h) に属するノードのいずれかから，メッセージ (以後 ρ型メッセージと呼ぶ) として受けとることにする．一方，更新されないレジスタ値の場合は，フェーズ (2) で送信依頼メッセージ (χ型メッセージ) を Snode(s_i) から送信し，それを受取ったノードは必要なレジスタ値を Snode(s_j) へρ型メッセージで送信する．

これにより，次の責任ノードでも (イ) で述べた仮定が満たされる (よって全ての責任ノードで (イ) の仮定が成り立つ)．

まとめると，フェーズ (2) で送信するメッセージはα型，η型，λ型，χ型の 4 つの型があり，場合によっては 2 つ以上の型のメッセージを同時に送る必要がある場合もある．この場合，それらのメッセージはフェーズ (2) で一つのメッセージとして送信する．このようにフェーズ (2) で送信されるメッセージを，2 つ以上の型の合成が生じたか生じなかったかにかかわらず，μ型メッセージと呼ぶことにする．同様にフェーズ (5) で送信するγ型，ρ型，θ型のメッセージを合わせてξ型メッセージと呼ぶことにする (**表 15**)．

以上が各ノードの動作の概略である．以下では，各ノードが上述の各フェーズでどのノードとどのようなタイプ，内容のメッセージを交換すればよいのか，及び，その際の交換メッセージの総数を出来るだけ少なくするにはどうすればよいのかについて具体的に説明する．

各動作仕様の動作

フェーズ (2)，(3) でそれぞれμ型メッセージ，β型メッセージを受信する PEの集合をμ，βで表す．集合μは集合α，η，λ，χ の和集合となる．ここで集合α，η，λ，χ は，それぞれ，α型メッセージ，η型メッセージ，λ型メッセージ，χ型メッセージを受信するノードの集合を表す．

表 15: メッセージタイプ

各メッセージは状態遷移の識別子を含む．これ以外の内容をあげる．

型	from	to	内容	意味
α	Snode(s_i)	集合 α		β型メッセージの送信依頼
μ η	Snode(s_i)	集合 η	レジスタ値 入力変数値	更新のためのレジスタ値， 入力変数の転送
λ	Snode(s_i)	集合 λ		更新指示
χ	Snode(s_i)	集合 χ		ρ型メッセージの送信依頼
β	集合 α	集合 β	レジスタ値	更新のためのレジスタ値の転送
γ	集合 γ	Snode(s_j)		更新終了通知
ξ ρ	集合 ρ	Snode(s_j)	レジスタ値	Snode(s_j) が次状態の 条件判定等に必要な レジスタ値の転送
θ	Snode(s_i)	Snode(s_j)		責任ノード交替

またフェーズ (5) で，γ型メッセージ，ρ型メッセージを送信するノードの集合をξで表す．集合ξは，γ型メッセージを送信する PEの集合γと，Snode(s_j) が必要とするレジスタ値を送る PEの集合ρの和集合となる[11]．なお，ノード集合γは，この遷移でレジスタ値の更新を行なうノードの集合となる (**表 16**)．

このとき，SSの各状態遷移に対して，各 PE_kは次のように状態遷移の系列を実行すると考える．

[動作系列 **TS**]

(Proc1). PE_k = Snode(s_i) ならば，PE_kは状態 s_iで各動作が実行可能かどうかを判定する．実行可能な動作があれば実行する．

(Proc2). PE_k = Snode(s_i) ならば，kから$\mu(=\alpha+\eta+\lambda+\chi)$ に属する PEにμ型のメッセージを送る．

$PE_k \in \mu$ならば PE_kが，Snode(s_i) からのμ型のメッセージを受け取る．

[11] メッセージの場合と異なり，θ型は除外している．

表 16: PEの集合

集合	集合に属するノード (PE) のもつ性質
α	更新に必要なレジスタの値を送るべき PE
β	更新に必要なレジスタの値を受け取る PE
γ	レジスタの値を更新すべき PE
η	Cset(s_i) に含まれるレジスタの値や入力変数値を Snode(s_i) から受け取るべき PE
λ	自ノードのレジスタを用いて 更新できる PE
ρ	Cset(s_j) に含まれるレジスタの値を Snode(s_j) に送るべき PE
χ	集合 $\rho - \gamma$ に含まれる PE
μ	集合 $\alpha \cup \eta \cup \lambda \cup \chi$ に含まれる PE
ξ	集合 $\rho \cup \gamma$ に含まれる PE

(Proc3). $PE_k \in \alpha$ならば PE_kからβに属する PE にレジスタ値 (β型のメッセージ)を送る.

$PE_k \in \beta$ならば PE_kがβ型メッセージを受け取る.

(Proc4). $PE_k \in \gamma$ならば PE_kはレジスタ値を更新する.

(Proc5). $PE_k \in \xi(=\gamma+\rho)$ ならば PE_kから Snode(s_j) にξ型のメッセージを送る.

$PE_k =$ Snode(s_j) ならば PE_kがξからのメッセージを受け取る.

θ型メッセージを送る必要がある場合は, Snode(s_i) から Snode(s_j) へ直接送る. □

上記 (Proc2), (Proc3), (Proc5)において, メッセージを受信する PEは, 受信の前に遷移条件を満たすかどうかを調べる.

もし, PE_kが **TS** のどの動作にも関与しなければ, $s_i \rightarrow s_j$はϵ遷移に置き換える. この方針に従って SS中の各 $s_i \rightarrow s_j$を **TS** に対応する動作系列に置き換えることにより PE_kが得られる.

4.3.2 送受信メッセージの決定法

上記の方法では, 3 ステップ (フェーズ (2),(3),(5)) でメッセージ交換が行われる. 勿論, 4 ステップ以上のメッセージ交換による方法など, 上記以外の導出方法も考えられる. これ

らの方法と比較して, 本方法はメッセージ総数が少ない, メッセージ交換によるシステム全体での遅延時間が比較的短い, 等の利点がある.

次にメッセージ数の削減について考える. 一般に, PE_kから PE'_kへ同期メッセージのみを送る場合と複数個のレジスタ値の組を送る場合ではメッセージ長に大きな違いがあるが, ここでは十分な転送速度が得られると仮定し, メッセージ長にかかわらず, 同一フェーズで各 PEに送られるμ,β,ξ型メッセージをそれぞれ単位と考え, その仮定の下でメッセージの総数を少なくする方法を考える.

ある PEがあるレジスタの値を必要とする場合, η型メッセージ, β型メッセージのいずれかを用いることが考えられる. 例えば PE_1が責任ノードで, PE_1, PE_3にレジスタ R_1があり, PE_2が R_1の値を必要とする場合, PE_1が PE_3にα型メッセージを送り, PE_3がβ型メッセージで R_1の値を PE_2へ送るよりは, PE_1が直接 PE_2へη型メッセージで R_1の値を送る方がメッセージ総数を少なくできる (前者は 2 個, 後者は 1 個). 一方, 今の例にレジスタ R_3を PE_3に加え, PE_2が R_1, R_3の値を必要とする場合を考える. この場合 PE_1から PE_3へα型メッセージを送り, β型メッセージを用いて R_1, R_3の組を PE_3から PE_2に送るとメッセージ総数は 2 個で済むが, PE_1から PE_2へη型メッセージでレジスタ R_1の値を送るとレジスタ R_3の値を送るためにさらにα型とβ型メッセージを発行する必要があり合計 3 個のメッセージが必要である.

このような様々な場合を考慮に入れ, できるだけ効率的なメッセージの送信方法を考える必要がある.

以下では, 各 PEが, どのタイプのメッセージで, どのレジスタ値の組を, どのノードと送受信しなければならないかを, 具体的にどのように決定するかについて述べる.

[定義 12] [メッセージの送信方法の決定に用いる命題変数]

Snode(s_i) を PE_uとする.

(I). 上の (Proc2)で PE_uから PE_vへη型メッセージを送らなければならないとき真となるような命題変数をηuvとする. また, PE_uから PE_vへレジスタ R_h (入力 x) の値を送信する必要があるとき真となる命題変数をηuv_R_h (ηuv_x) とする.

(II). 上の (Proc2)で PE_uから PE_vへα型メッセージを送る必要があるとき真となるような命題変数をαuvとする.

(III). 上の (Proc2)で PE_uから PE_vへλ型メッセージを送る必要があるとき真となる命題変数をλuvとする.

(IV). 上の (Proc5)で PE_uから PE_vへχ型メッセージを送る必要があるとき真となる命題変数をχuvとする.

(V). 上の (Proc2)で$\eta uv, \alpha uv, \lambda uv, \chi uv$の何れかが真であるとき真となる命題変数を$\mu uv$とする.

(VI). 上の (Proc3)でαに属するノード wからβに属する PE_vにレジスタ値を送る必要があるとき真となる命題変数をβwvとする. また, PE_wからノード PE_vへレジスタ R_hの値を送信する必要があるとき真となる命題変数をβwv_R_hとする.

(VII). 上の (Proc5)でξに属する PE_vからノード Snode (s_j) (以下 PE_zとする) にγ型メッセージ, ρ型メッセージを送る必要があるとき真となる命題変数をそれぞれ, γvz, ρvz とする. このうち, レジスタ R_h の値を送る必要があるとき真となる命題変数をρvz_R_hとする. また, ξが空集合のとき真となる命題変数をθuzとする. また, $\gamma vz, \rho vz, \theta vz$の少なくとも一つが真のとき真となる命題変数を$\xi vz$とする. □

各命題変数の決定に用いる制約条件

定義12の (I)~(VII)の命題変数 ηuv, ηuv_R_h, ηuv_x, αuv, λuv, χuv, μuv, βwv, βwv_R_h, ξvz, γvz, ρvz, ρvz_R_h, θuzの間に次の不等式が成り立つ必要がある (ここでは各命題変数を 0, 1 のいずれかの値を取る整数型の変数とみなす).

以下, u を Snode(s_i) とする. また zを Snode(s_j) とする.

(1) $R_h \in$ Cset(s_i) なる各レジスタ R_hと $PE_v(1 \leq v \leq p)$ の組に対して,

$$\eta uv \geq \eta uv\_R_h$$

(2) 入力 x と $PE_v(1 \leq v \leq p)$ の組に対して，

$$\eta uv \geq \eta uv\_x$$

(3) PE_vが入力 x を必要とするとき (PE_vのレジスタ値の更新に入力 x の値が必要なとき)

$$\eta uv\_x = 1 \qquad \text{上述の } (I) \text{に対応}$$

(4) 各 $PE_v(1 \leq v \leq p)$ に対して，

$$\mu uv \geq \alpha uv, \mu uv \geq \eta uv,$$

$$\mu uv \geq \lambda uv, \mu uv \geq \chi uv \qquad \text{上述の } (V) \text{に対応}$$

(5) 各 $PE_v(1 \leq v \leq p)$ と $w(1 \leq w \leq p)$ の組に対して，

$$\alpha uv \geq \beta vw \qquad \text{上述の } (II) \text{に対応}$$

これはβ型のメッセージを送信するノード vにはノード u からα型のメッセージを送らなければならないことを表している．

(6) 各 $PE_v(1 \leq v \leq p)$ とレジスタ $R_h(1 \leq h \leq m)$ の組に対して，もしノード vが R_hの値を必要とするなら，

$$\sum_{w \in \mathrm{Rnode}(R_h)} \beta wv\_R_h + \eta uv\_R_h \geq 1 : R_h \in \mathrm{Cset}(s_i)$$

$$\sum_{w \in \mathrm{Rnode}(R_h)} \beta wv\_R_h \geq 1 : R_h \notin \mathrm{Cset}(s_i)$$

これはレジスタ R_hの値をβ型, η型のいずれのメッセージで受けとっても良いことを表している．

(7) また，各 $PE_v(1 \leq v \leq p)$ と $w(1 \leq w \leq p)$ の組に対して，

$$\beta wv \geq \beta wv\_R_1, \ldots, \beta wv \geq \beta wv\_R_m \qquad \text{上述の } (VI) \text{に対応}$$

(8) 各 $PE_v \in \lambda$に対して，

$$\lambda uv \geq 1 \qquad \text{上述の } (III) \text{に対応}$$

自力でレジスタ値を更新できるノード vは λ型のメッセージを受けとらなければいけないことを表している.

(9) 各 $PE_v \in \gamma$に対して,

$$\gamma vz = 1$$

(10) また, 各レジスタ $R_h(1 \leq h \leq m)$ に対して, もし PE_zが R_hの値を必要とする $(z \notin \mathrm{Rnode}(R_h))$ なら,

$$\sum_{w \in \mathrm{Rnode}(R_h)} \rho wz\_R_h \geq 1$$

この式では, ρ型メッセージを, 実際にレジスタを持っているところからのみ受けとることにしている. $\mathrm{Snode}(s_i)$ は $\mathrm{Cset}(s_i)$ として, 実際に自ノードが持っている以上のレジスタを知っている可能性があるがここではそれは取り扱わない.

(11) さらに, 各 $PE_w(1 \leq w \leq p)$ とレジスタ $R_h(1 \leq h \leq m)$ の組に対して,

$$\rho wz \geq \rho wz\_R_h$$

$$\xi wz \geq \rho wz, \xi wz \geq \gamma wz, \xi wz \geq \theta wz, \qquad \text{上述の } (VII) \text{に対応}$$

(12) 各 $PE_v \notin \gamma$に対して,

$$\chi uv \geq \rho vz \qquad \text{上述の } (IV) \text{に対応}$$

これはレジスタの更新は行わないがρ型のメッセージを送らなければならないノード vには, ノード u からχ型のメッセージを送らなければならないことを表している.

(13) ξが空集合のとき,

$$\theta uz = 1$$

これは, PE間のメッセージのやりとりが全くなく, しかも責任ノードが替わる場合, PE_uから直接 PE_zへθ型のメッセージを送る必要があることを表している.

各変数の決定の方法

上述の (1)~(13) の制約条件を満足するような変数の組の中で,

$$N = \left\{ \sum_{1 \leq y \leq p \wedge y \neq u} \mu uy \right\} + \left\{ \sum_{1 \leq w \leq p \wedge w \neq u} \sum_{1 \leq v \leq p} \beta wv \right\} + \left\{ \sum_{1 \leq y \leq p \wedge y \neq z} \xi yz \right\}$$

の値を最小にするように各変数の値を定めれば，メッセージの総数が小さくなる．ここでNは，この状態遷移において，p個のPE間で交換されるメッセージの総数を表している．なお，各和演算では，自ノードへのメッセージの送信を行うような変数を排除している．

制約条件は線形不等式であるので，Nを目的関数として，0-1 整数線形計画問題を解くアルゴリズムを用いてNの値を最小にするような解を求めればよい．求めた解から各PE_kがどのタイミングでどのレジスタ値をどのPEと送受信しなければならないかがわかる．なお，0-1 整数線形計画問題は整数線形計画問題の制約式の各変数値を 0,1 に限定したもので，一般には NP 完全であるが[31]，実用的には様々な高速化の手法が考案されている．また多くの場合，レジスタの数は十数個程度，制約式の数も百個程度に抑えられると考えられるので，現実に計算可能であるといえる．

4.3.3 各動作仕様の構成方法

まずPE間で交換するメッセージの内容について説明する．ついで，各動作仕様の構成方法について述べる．

各状態遷移$e(s_i \rightarrow s_j)$について，$PE_u(=\mathrm{Snode}(s_i))$から$PE_v(1 \leq v \leq p)$へ送信する$\mu$型のメッセージの内容は次のとおり：

$$\langle \mu, \mathrm{label}(e), \{\langle \text{“R}_\text{h}\text{”}, \mathrm{value}(R_h)\rangle \mid \eta uv\_R_h = 1\}, \{\langle \text{“x”}, \mathrm{value}(x)\rangle \mid \eta uv\_x = 1\}\rangle$$

但し，label(e)は辺eを他辺と区別するためのラベル，value(R_h), value(x)はそれぞれ，レジスタR_h, 入力xの値である．μ型メッセージは，$\eta, \alpha, \chi, \lambda$型メッセージの内容をあわせているが，$\alpha, \chi, \lambda$型メッセージについては，内容は，タイプ，label(e)があればよい．η型メッセージのためにレジスタ値等のデータが追加される．どのノードvにどのレジスタを送信すべきかは，変数ηuv_R_hの値から判定できるので，これが 1 であるレジスタ値のみ送ればよい．

同様に，PE_vから$PE_w(1 \leq v, w \leq p)$へ送信するβ型のメッセージの内容を

$$\langle \beta, \mathrm{label}(e), \{\langle \text{“R}_\text{h}\text{”}, \mathrm{value}(R_h)\rangle \mid \beta uv\_R_h = 1\}\rangle$$

とし，$PE_v(1 \leq v \leq p)$から$PE_z(=\mathrm{Snode}(s_j))$へ送信する$\xi$型のメッセージの内容を

$$\langle \xi, \mathrm{label}(e), \{\langle \text{“R}_\text{h}\text{”}, \mathrm{value}(R_h)\rangle \mid \rho uv\_R_h = 1\}\rangle$$

とする．すなわち，各メッセージにはタイプ，状態遷移 e のラベル名及び，送信すべきレジスタ名 (入力変数名) とその値が含まれている．

責任ノードでない場合は，責任ノードがどの状態遷移を実行したかがわからないので，メッセージに含まれる状態遷移 e のラベル名から状態遷移を判断する．なお 図8では $\langle \eta, 2, \{\langle$"$R_1$", value$(R_1)\rangle\}\rangle$ を$\eta^2, \{\langle R_1, v_1\rangle\}$ と略記している．また遷移条件 $w = \eta^2$は入力メッセージのタイプがη型でそのラベルが 2 であるとき真となる．

動作仕様群の構成法

各 PEの状態遷移を 上記で得られた述語の値にしたがって，動作系列 **TS** に置き換えれば各 PEの状態遷移系列を生成できる．もちろんこの構成法は，述語の値が正しく求まる限り停止する．ここでは，状態遷移の各ラベルの与え方を中心にやや詳しく述べる．各ノードには σで割り当てられたレジスタに加え，作業用のレジスタ R_0を割り当てる．

(Proc1)では，PE_kが Snode(s_i) であるとき，全体仕様の遷移条件，動作をそのまま記述する (レジスタ更新は行なわない．これは (Proc4)でまとめて行なう)．但し，Snode(s_i) である PEが外部入力 x の受信動作をした場合，x の値をレジスタ R_0に追加するレジスタ更新式 $R_0 \leftarrow$put(R_0, x) を与える．

(Proc2)では，PE_kが Snode(s_i) ならμ型のメッセージを (複数) 出力する必要があるが，このための動作は適当な順序で直列に行なえばよい．遷移条件は常に真である．各メッセージのタイプはμであり．ラベルは選択された状態遷移が $e(s_i \rightarrow s_j)$ のとき，label(e) となる．kが Snode(s_i) 以外の PEで，μ型のメッセージ m を受信する必要のある PEなら，状態遷移 e に対応する状態遷移系列の遷移条件 として $m = \mu^{\text{label}(e)}$を与える．動作として，$PE_u$ (= Snode(s_i)) からのメッセージ m の入力 $g_{uk}?m$, レジスタ更新式として，$R_0 \leftarrow$put(R_0,m) を与える．(Proc3)で，複数のβ型のメッセージの送信を行なう必要のある $PE_v(\in \alpha)$ は (Proc2)と同様に適当な順序で直列に行なう．また，β型のメッセージを受信する必要のある $PE_k(\in \beta)$ は，それらのメッセージの受信動作を行なう．kの動作として，PE_v $(\in \alpha)$ からのメッセージ m の入力 $g_{vk}?m$, レジスタ更新式として，$R_0 \leftarrow$put(R_0,m) を与える．受信動作も適当な順序で直列に行なえばよい (受信メッセージの到着順は一意に定まらないが同一

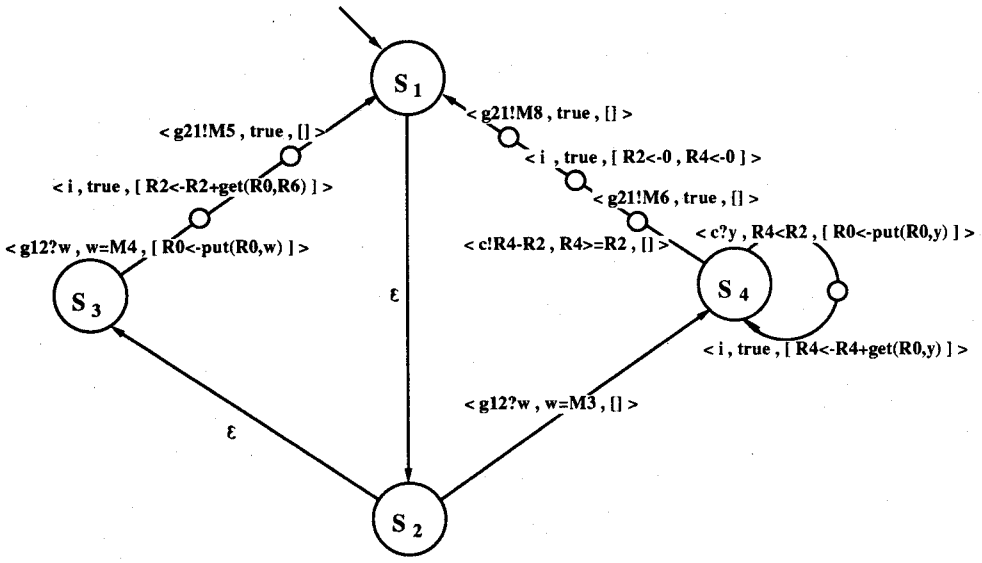


図 9: 動作仕様 PE_2(状態簡約前)

Fig.9 PE_2 with ϵ Moves

ゲートからは，同一タイプのメッセージは高々1 個しか受信しないので受信動作の順序を適当に定めても問題は生じない.)．遷移条件として $m = \beta^{\mathrm{label}(e)}$ を与えておく．(Proc4)で レジスタ R_0から必要な入力レジスタの値を $\mathrm{get}(R_0, R_h)$, $\mathrm{get}(R_0, x)$ を用いて取り出し，レジスタ更新を行なう．(Proc5)では，PE_kがξに属していれば，$\mathrm{Snode}(s_j)$ にξ型のメッセージを送信する．PE_kが $\mathrm{Snode}(s_j)$ なら，ξ型メッセージの受信動作を先ほどと同様に行なう．

状態簡約

上述の手続きによって導出される各 PE_kの状態遷移図は，いわゆるϵ遷移を含んでいることがある．例えば，図 **8**の $S1$ から上述の手順によって導出される PE_2は図 **9**のようになる．

全体仕様にある状態は各 PE_kにおいて，責任ノードとなる状態とならない状態に分けることができる．各 PE_kの作り方より，PE_kが責任ノードである状態 s と責任ノードでない状態 s'間にϵ遷移がないこと及び，責任ノードでない状態からの遷移は他 PEからのメッセージの受信動作で始まることが保証できる．そこで，各 PE_kの状態の中で，責任ノードでない状態 (図 **9**では S_1, S_2, S_3) をすべて一つの状態 (例えば S_0とする) にまとめ，ϵ遷移を取り

除くことにより状態簡約を行なう．図 9の PE_2から状態簡約を行なうと図 8の PE_2が得られる．なお受信メッセージには状態遷移のラベルがついているので状態簡約で複数の状態を一つにまとめても受信メッセージの内容からもとの状態遷移図中のどの遷移が選ばれたかを区別することができる．よって簡約後の PE_kは簡約前の PE_kと同じ動作が行なえる．

4.4 評価

1 状態遷移あたりのメッセージの決定に要する時間を News5000 (100 misp) で計測した．ノード数 5 レジスタ数 10 を平均とし，種々の状態遷移をランダムに与えた．この結果，全体の 1%が 1 秒以内，99%が 5 分以内で導出できることがわかった．これにより，この程度の規模であれば十分な値と考えられる．

なお，ヒューリスティックアルゴリズムも用意している．メッセージの平均総数は最適解の 1.1 倍程度 であり，すべて 1 秒以内で導出している．

動作仕様群の構成法の正当性

[定理 1]

上述の導出法で導出される分散システムの動作仕様 PE^{1-p}と全体仕様 SSの等価性が成立つ[37]． □

証明: まず，(1)．各状態遷移に対して，状態簡約前において動作仕様群と全体仕様の等価性が成立つことを示す．次に，(2)．簡約化アルゴリズムを適用した動作仕様群について，適用前の動作仕様と等価性がなりたつことを示す．

(1)．全体仕様 SSが初期状態にあるとき，動作仕様の PE^{1-p}はそれぞれの初期状態にあり，すべてのキューは空である．また，この状態における実行ノードが一つあり，遷移条件，動作，レジスタ値ともに全体仕様 SSに等しいことは構成法よりあきらか．

全体仕様 SSが状態 S_iにあるとき，PE^{1-p}がそれぞれ状態 S_iにあり，すべてのキューは空とする．PE_qが Snode(S_i) であるとする．PE_qの状態 S_iでは，遷移条件，動作ともに SSに等しい．また，PE_q以外の PE_kでは，状態 S_iからの遷移は，受信動作で始まるか，ϵ遷移であることは，構成法より簡単に分かる．

PE_qが状態S_iからの遷移を一つ選んだとき, PE_q以外のノードが, どの遷移を選べばよいかは, メッセージの構成法より, label(e) を調べればわかる. PE^{1-p}がこの遷移により, 正しくレジスタを更新することは, 構成法より明らか.

遷移後, SSの状態がS_jであれば, PE_q及び, 何等かの受信動作が行なわれたPE_kは状態S_jにある. その他の$PE_{k'}$では, 次に行なえる動作は結局, 受信動作しかないのでそのPEはどの状態にあるか分かっていると考えてよい.

またこの一連のメッセージの送受信は1対1対応しており, また Snode(S_j) は, 変更のあったすべてのPEからのメッセージを受け取った後でないと状態S_jに遷移できないので, S_jに遷移した後はすべてのキューは空である.

(2). この (1) で得られた任意のPE_kにおいて, PE_kが責任ノードである状態Sと責任ノードでない状態S'間にϵ遷移がないことが保証できるので, 責任ノードでない状態を前節のアルゴリズムで一つの状態にまとめても, もとのPE_kと同じ動作が行なえることを確認すれば十分である.

もとのPE_kにおける異なる2つの状態S', S''がこのアルゴリズムにより, 一つの状態Sになったとする. アルゴリズムより, PE_kは状態S', S''の実行ノードではないので, S', S''からの遷移はすべて受信動作で始まる. したがってSからの遷移もすべて受信動作で始まる.

Snode(S'') によって選ばれた遷移でPE_kが受信動作を行なうのであれば, メッセージ内のラベル label(e) によって, 状態Sからもともと状態S''にあった対応する遷移を行ない, 異なる遷移をすることはない. 受信動作を行なわないときは, PE_kの動作はϵ遷移であり, PE_kは何もしなくてよい. □

4.5 結言

本章では, 分散システムの全体仕様から動作仕様群を自動生成するアルゴリズムを与えた. 本章のモデルでは, 非決定性の動作が記述できる. また, 各リソースをレジスタとして表し, 入力と有限制御部の状態だけでなくその時点のレジスタ値に依存して次の状態やレジスタ値を定めることができる. レジスタのデータ型や次のレジスタ値を定める関数は設計者が自由に記述でき, OSIプロトコルなど多くの分散システムの仕様をこのクラスで記述できる.

5 リンク故障を起こす通信環境における分散実行動作仕様群の自動導出

5.1 序言

通信ネットワークのリンク，ノード等に故障を仮定して，それに対しても全体としてシステムの動作が保証されるフォールトトレランスに関する研究が多くなされている．リンク故障に関しては，代替経路を保証するために，例えばルーティングを複数用意しておき，それらから動的に適応可能なルーティングを選択する方法などが知られている．また，プロトコル合成の際に，エラーリカバリ性を持たすような研究も報告されている[21, 32, 27]．一般に，動的な経路制御は複雑であり，CPU 時間，通信路等の多くの資源を要する．また，下位層のプロトコルなどでは acknowledge メッセージを用いて経路の状態を調べる方法が一般的に用いられているが，この方法ではタイムアウト機構が必要である．一般に タイムアウトの時間は各動作時間に比べてきわめて大きな値が設定されるため，タイムアウト機構が働くと全体の処理効率が著しく低下する．このため，アプリケーションのレベルではこのような制御をせず，最初からメッセージを多重化して送信する方法が考えられる．

そこで本章では，高々一ヶ所のリンクの故障が発生しても全体仕様通りに動くようにメッセージ交換を行う動作仕様を導出する．耐故障性のため，同一メッセージを異なる二経路で送受信するが，単純に二重化したのでは，メッセージ交換の回数が多くなる．そこで，メッセージ交換に用いるリンクの数をできるだけ少なくしたり，同一リンクを同じタイミングで送られる複数のメッセージを一つに統合すること等により，提案する模倣方針のもとで，各状態遷移でのノード間の送受信動作の総数が最小になるように各ノードの動作仕様を導出する．

5.2 リンク故障を許す分散実行環境

この章であつかう分散環境をここで述べる．PE_iから PE_jへの通信路 (リンク) は無限容量を持つ信頼性の保証されていないリンク (buff_{ij}) で結ばれているとする[12]．

リンクの故障はメッセージ消失のみとし，以下の性質を仮定する．

[12] i から jへのリンクと，jから iへのリンクは区別する

- もし，リンクが故障していなければ，そのリンクはFIFOキューとして動作する．
- リンクが故障状態になれば，そのリンク内のすべてのメッセージは消失する．また，故障状態のリンクに入ったメッセージも消失する．
- 任意の時点で高々一ヶ所のリンクしか故障状態にならない．
- あるリンクが故障状態から復帰すれば，そこからある単位期間内はいずれのリンクも故障しない．

ここでは，メッセージの消失のみを取り扱う．伝送路の雑音によりメッセージの一部が欠落，変質することも，実際問題として起こりうる．この場合，メッセージにチェックサムをつけることにより，データの一部変質や欠落は検出可能である．もし，異なるメッセージを受信した場合，どちらのメッセージが間違ったものなのかはチェックサムを見ることにより判定できる．これにより，間違ったチェックサムの方を破棄してメッセージの消失として取り扱えばよい．もちろん，両方のチェックサムが間違っていた場合は，両方のメッセージが消失したものとみなすので，本稿の仮定である高々1つのリンクエラーという仮定を満たさなくなる．

なお，リンク中のメッセージの最大伝送処理時間，レジスタ更新に要する最大時間をそれぞれ，Δ，δとし，その他動作に要する時間を無視できるものと仮定し，$4\Delta+\delta$時間を上述の単位時間 (すなわち，リンク故障から次のリンク故障までの最低時間) とする．この値は全体仕様の一つの状態遷移を (p 個の) 全ノードで実行するのに必要な時間の最大値に相当する．詳細については，**5.3.1節**で述べる．

この章では**条件2**の他にさらに以下の制約を設ける．

[条件 3]

(1). ノードの数 p は3以上であること．

(2). σにおいて各レジスタは2個以上のノードに分散配置されていること． □

この章で用いる全体仕様 $S2$ と割り当てσを以下のように定める図 **10**．また，このときの責任ノード (PE) 等は以下のように定まる．

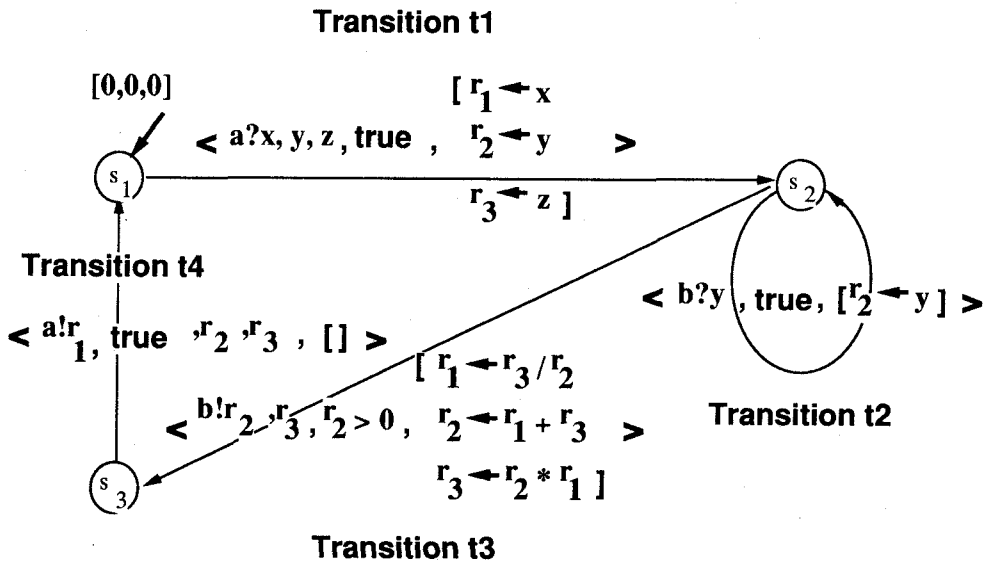


図 10: 全体仕様の例 $S2$

Fig.10 Service Specification $SS2$

	PE_1	PE_2	PE_3	PE_4
レジスタ	r_1	r_2, r_3	r_1, r_3	r_2, r_3
ゲート	a	b		

Snode $\mathrm{Snode}(s_1) \equiv PE_1$, $\mathrm{Snode}(s_2) \equiv PE_2$, $\mathrm{Snode}(s_3) \equiv PE_1$,

Rnode $\mathrm{Rnode}(r_1) \equiv \{PE_1, PE_3\}$, $\mathrm{Rnode}(r_2) \equiv \{PE_2, PE_4\}$, $\mathrm{Rnode}(r_3) \equiv \{PE_2, PE_3, PE_4\}$

Cset $\mathrm{Cset}(s_1) \equiv \{r_1\}$, $\mathrm{Cset}(s_2) \equiv \{r_2, r_3\}$, $\mathrm{Cset}(s_3) \equiv \{r_1, r_2, r_3\}$ □

[例 5]

全体仕様 $S2$(図 **10**) と割り当て σが与えられたとき，一つの状態遷移 $s_2 - \langle b!r_2, r_3, r_2 > 0, [r_1 \leftarrow r_2/r_2,\ r_2 \leftarrow r_1 + r_3,\ r_3 \leftarrow r_2 \times r_1] \rangle \rightarrow s_3$ に対する各動作仕様 PE^{1-4} の一つの可能なタイミングチャートを 図 **11**に表す.

図 11において，まず PE_2 が 出力動作 $b!r_2, r_3$ を 選択し，実行する．ステージ (I) では，レジスタ更新のために必要なメッセージの送受信が各 PE_k間で行われ，ステージ (II) では，レジスタ更新を行ったすべての PE_kが PE_1 に更新終了を表すメッセージや以降 PE_1が必要とするレジスタ値 (r_2, r_3) などを含むメッセージを送信する．PE_1 はメッセージを適宜，

受信した後次の遷移に対応する動作を実行する (各メッセージには，状態遷移名の情報があるとする).

ここで，ステージ (I) では，いずれの レジスタ値も異なった経路で二重化されて送受信されるので， 受信側で二度目に受信したメッセージは，もし，すべてのリンクが信頼できるのであれば，無視できる．**図 11**では，破線で表されるメッセージは，二度目の受信となるので無視されるメッセージを表している．

ところが，リンク故障がある場合は振る舞いは変わってくる．例として，PE_3 から PE_4 のリンク $\mathbf{buff}_{34}$が故障していると仮定する．このとき レジスタ r_1 の情報は PE_3 から PE_4 には送られない．しかし，レジスタ r_1 の情報は PE_1からは伝えられる．よって，リンク $\mathbf{buff}_{34}$が故障していても，PE_4 は自ノードのレジスタ値を更新することが出来る．**図 11**では，どのリンクがそのように故障しても，すべての PE_kが自ノードのレジスタ値を更新することが可能である．また同様に，PE_1 は，すべての PE_kがレジスタ更新を終了し，ステージ (II) が終了したことが分かる．よって， **図 11** は，高々1 つのリンクが故障する限りにおいて，正しく動作することがわかる．同様の 動作系列を他の状態遷移でも導出することが可能である．これらの系列を接続することにより，全体仕様通りに動く動作仕様を導出することが出来る．その動作仕様 $\langle PE_1, \ldots, PE_4 \rangle$ の例を付録に付す． □

5.3 リンク故障を考慮に入れた動作仕様群の導出

この節では, リンク故障を考慮に入れた動作仕様群の導出のための基本方針, メッセージの決定法等について述べる.

5.3.1 基本方針

ここでは，メッセージの二重化を用いて故障に対応することとし，また，一状態遷移ごとにそれを模倣する各ノードの状態遷移系列を求めることにする．そこで，以下の部分問題 [SIM] を考える．そこで用いている模倣方針の妥当性については，**5.4節**で述べる．

部分問題 [SIM]

入力: 割り当てσ, リンク故障を考慮していない SS 中の 1 状態遷移 t

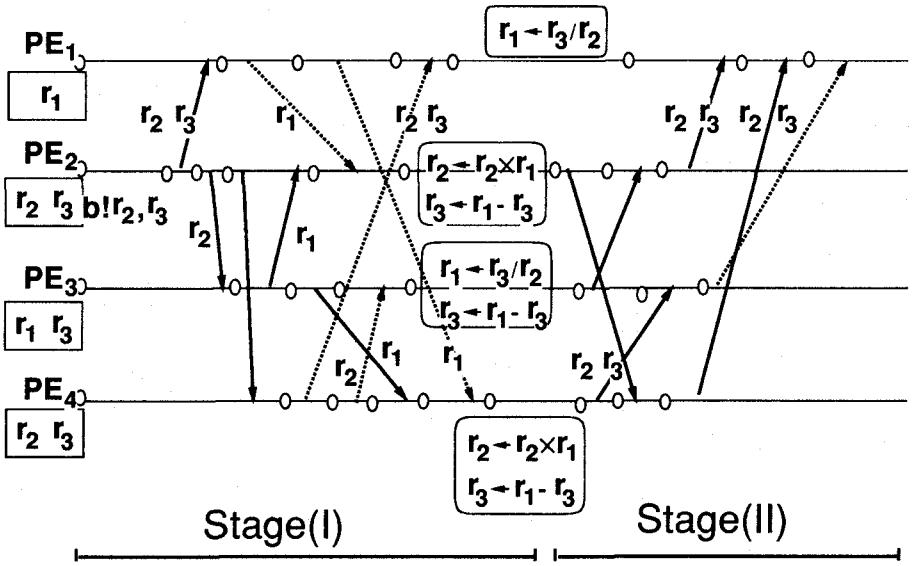


図 11: 遷移 $s_2 \rightarrow s_3$のタイミングチャート

Fig.11 A Timing Chart of Transition $s_2 \rightarrow s_3$

出力: 各 PE_kにおける，t を模倣するための状態遷移系列とメッセージ内容

条件: 後述の模倣方針のもとで (t を模倣する間の) 総メッセージ数を最小にすること． □

[模倣方針] 全体仕様の 1 状態遷移を次の 2 ステージで実現する．

ステージ (I) レジスタ更新までのメッセージ転送

ステージ (I) を以下のフェーズで構成する．

(1) 責任ノードが実行可能な遷移を決め，動作を実行する．

(2) 責任ノードがメッセージを送信し，中継すべきノードがこれを受信する．

(3) 中継ノードがメッセージを中継し，最終的な受信ノードがこれを受取る．

(4) レジスタを更新すべきノードがレジスタを更新する．

図 12(I) はフェーズ (2)，(3) におけるメッセージ転送を表したものであり，**条件 3** より，3 つの場合を考えれば十分である．(a) は更新を行うべきノード (二重丸) が更新に必要とするレジスタの一つに着目したとき，そのレジスタが自ノードと責任ノード (×丸) 以外の二ヶ所以上のノード (網がけ丸) にある場合である．責任ノードがレジスタを持っている 2 つの

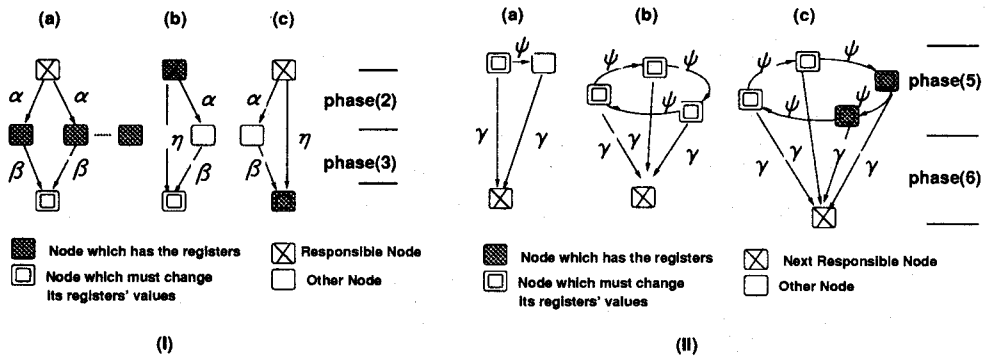


図 12: 耐故障を考慮に入れたメッセージ交換

Fig.12 Exchange of Messages

ノードにきっかけを与えるメッセージαを送り，これを受け取ったノードが，そのレジスタ値を用いてレジスタ更新すべきノードに，(レジスタの値の情報を持つ) メッセージβを送ることを表している．次にレジスタを持つノードの一つが責任ノードである場合，(a) のように，2 つのノード中継ノードとしてレジスタ値を送信してもよいし，(b) のように，一方のメッセージηを直接送ることも考えれられる．(a),(b) のうち，メッセージ数の少ない方を用いるが，この詳細は 5.3.2節で述べる．(c) は，自ノードでレジスタ値を更新できる場合である．この場合，更新のきっかけを与えるメッセージを二重化して送る．この場合もα,βメッセージを用いるが，レジスタ値の情報は持たなくてよい．いずれの場合でも，いずれか一つのメッセージが消失してもレジスタ更新すべきノードがレジスタ更新を行うことができる．また，責任ノードが フェーズ (1) で自ノードのレジスタ値をあらかじめ更新することも考えられる．これが可能な場合はこれを行う．

ステージ **(II)** 更新終了通知のメッセージ転送

図 12(II)(a) は，更新を行うノードが 1 つの場合である．この場合，更新終了したノードが更新終了メッセージを次の責任ノードに送信するときメッセージの二重化のため，3 つのメッセージが必要である．一般に更新を行うノードは複数存在すると考えられる．このとき (b) のようなメッセージの送受信パターンとする．まず，更新を行ったすべてのノード間でサイクルを形成するように更新終了メッセージψを 1 つだけとなりへ一斉に送信する．次

に各ノードは，ψを受信したなら，他ノードから受取った更新終了情報と自ノードの更新終了情報の二つの情報をまとめて次の責任ノードに送る (メッセージγ)．このようにすれば，高々1つのリンク故障が生じても，次の責任ノードはすべての更新ノードが更新終了したことを知ることが出来る．**図 12(II)** のように，ステージ (II) の2つの部分をそれぞれフェーズ (5)，(6) とする．

この際，次の責任ノードが必要とするレジスタ値を送信できるノードが，レジスタ更新をしたノード集合中にあれば，(c) のように，その一つがレジスタ値を，次の責任ノードに送信する．次責任ノードが必要とするレジスタの中には，この遷移で更新されないものがあるかも知れない．その場合は，フェーズ (I) において，**図 12(I)** (a)，(b) の送受信パターンを用いて，あらかじめ，現在の責任ノードから次責任ノードへ必要なレジスタ値を送信しておくことにする．これは**図 12(I)**(a)，(b) の二重丸で表しているノードを次責任ノードと読み変えれば良い．

なお，**5.2節** の単位時間 $4\Delta+\delta$については，上述の方法では，一状態遷移を模倣する系列中に最大4回メッセージの送受信が行われる個所しかないこと，レジスタ更新も高々1回であるため，$4\Delta+\delta$時間経過すれば必ず次の状態遷移に移行しており，全体仕様での状態遷移を実行中に高々一つのリンク故障しか発生しないという条件が満たされる．

このメッセージ交換法は，全体仕様の一状態遷移を各動作仕様が模倣する際に，なるべく少ないフェーズでメッセージ交換することを意図している．

古いメッセージの処理について

前節で述べた方法では，二重化されたメッセージの一方のみを受信することにより，次の動作に進むため，古い (後で到着した) メッセージを矛盾の生じないようにうまく破棄する必要がある．本論文では，“タイムスタンプ”と呼ばれる識別子を用いてこのことを実現する．

タイムスタンプはノード間でやり取りされるすべてのメッセージに付加する．その値は，初期状態では0で，各責任ノードがステージ (I) の最初の動作を実行する毎に1ずつ増加させる．また，レジスタを更新したノードがステージ (II) の最初のメッセージを送信する際に

もその値を 1 ずつ増加させる．このようにタイムスタンプは，全体仕様 SSの 1 状態遷移の 1 ステージが終了するごとに 1 ずつ増えるようにしておく (すなわち 1 状態遷移 で 2 ずつ増える)．これを行うために，各 PE_kにはカウンタレジスタ r_cを持たせておく．また，メッセージ受信時には，r_cを (その時点で分かりうる) 最新のタイムスタンプにセットする．これにより，二重化したメッセージの一方のみを受信して，次のステージに進んだ場合や，一方のメッセージがステージの終了後に到着した場合でも，メッセージのタイムスタンプが r_cより小さければ破棄して良いことが分かり (一世代) 古いメッセージによる誤動作を防ぐことが出来る．このようなメッセージ破棄のための状態遷移があっても，観測合同のもとでは，等価性は保証できる．

5.3.2 送受信メッセージの決定法

以下では，前述の [模倣方針] とメッセージ統合の方法のもとで，1 状態遷移の模倣に必要な総メッセージ数を最小にするメッセージの送受信内容 (どの PEからどの PEへどの内容のメッセージを送るか) を決定する方法について述べる．例えば，**図 12(I)** の (a) の場合，中継ノードにどの 2 つを用いればメッセージ総数が少なくなるかは，他のレジスタ値の送信パターンに依存するので自明でない．同様に，責任ノードがレジスタ R_hを保持しており，そのレジスタ値をあるノードが必要とするときに，**図 12(I)** の (a)，(b) のいずれのパターンを用いるべきかも，同様の理由により自明でない．そこで，全体仕様 SSの各状態遷移について独立に，以下に述べる 0-1 整数線形不等式を導出し，これに対してメッセージ総数を表す目的関数の最小解を求めるという方法を用いる．最小解を与える変数の値から，どのノードからどのノードへどのタイミングでどのような内容のメッセージを送信すべきかが分かる．

ある状態遷移 $s_i - \langle a \cdots, C(\ldots), CR\rangle \rightarrow s_j$に対して，レジスタ更新する PEの集合 γ，自力でレジスタ更新できる PEの集合 λ，レジスタ更新の際入力変数 x を必要とする PEの集合 $\theta(x)$ は，SSと，σからただちに求めることができる．以下，状態 s_iの責任ノード $\mathrm{Snode}(s_i)$ を PE_uとする．また $\mathrm{Snode}(s_j)$ を PE_zとする．βwv を PE_wから PE_vへβ型メッセージを送信すべきときに 1，それ以外のとき 0 をとる命題変数とする．また，βwv_R_h

(βwv_x) を PE_wから PE_vへβ型メッセージでレジスタ R_h(入力変数 x) の値を送受信すべきときに 1，それ以外のとき 0 をとる命題変数とする．他の変数も同様とする．

各命題変数の決定に用いる制約条件

以下の式は (1)~(5) がステージ (I) に相当し，(6)~(9) がステージ (II) に相当する．

(1) もし，PE_vが，更新時にレジスタ R_hを必要とするなら，

(1-1)$R_h \notin \mathrm{Cset}(s_i)$ のとき

$$\sum_{w \in \mathrm{Rnode}(R_h)} \beta wv\_R_h \geq 2$$

(1-2)$R_h \in \mathrm{Cset}(s_i)$ のとき

$$\sum_{w \neq u \wedge 1 \leq w \leq p} \beta wv\_R_h + \eta uv\_R_h \geq 2$$

これは，PE_vがレジスタ R_hの値を 2 個所から受信することを表している．それぞれ，図 **12(I)**(a), (b) に対応する．**条件 3**より，この不等式を満たす解領域は存在する．

(2) 入力変数 x と各 $PE_v \in \theta(x)$ の組について，

$$\sum_{w \neq u \wedge 1 \leq w \leq p} \beta wv\_x + \eta uv\_x \geq 2$$

これは，PE_vが二ヶ所から入力変数の値を受信することを表している．

(3) 各 $PE_v \in \lambda$に対して，

$$\eta uv = 1, \qquad \sum_{1 \leq w \leq p \wedge w \neq u \wedge w \neq v} \beta wv \geq 1$$

これは，図 **12(I)**(c) に対応する．

(4) 任意の $PE_s, PE_t (1 \leq s \leq p,\ 1 \leq t \leq p)$ とレジスタ R_hの組について，

(4-1)$s \in \mathrm{Rnode}(R_h)$ のとき

$$\alpha us \geq \beta st\_R_h$$

(4-2)$s \notin \mathrm{Rnode}(R_h)$ のとき

$$\alpha us\_R_h \geq \beta st\_R_h$$

(5) 任意の $PE_s, PE_t (1 \leq s \leq p,\ 1 \leq t \leq p)$ と入力変数 x の組について，

$$\alpha us\_x \geq \beta st\_x, \quad \alpha us \geq \beta st$$

(4),(5) は，β型メッセージを送信する PE_s に対して必ず，PE_u からα型メッセージを送信すべきことを表している

次に，更新終了の通知，PE_zへの $\mathrm{Cset}(s_j)$ のレジスタ値の通知について考える．

(6) 各 $PE_v \in \gamma - \{z\}$ に対して，

$$\gamma vz = 1$$

(7) PE_zがレジスタ R_hを必要とするとき $(R_h \in \mathrm{Cset}(s_j) \wedge z \notin \mathrm{Rnode}(R_h))$，$R_h$と zの各組に対し，

(7-1) レジスタ R_hをもつノード wがレジスタ更新をする場合 $(PE_w \in \mathrm{Rnode}(R_h) \wedge PE_w \in \gamma)$，そのような，$w$を一つ選び，

$$\gamma wz\_R_h = 1$$

(7-2) レジスタ R_hをもつ PE_wがレジスタ更新をしない場合 $(PE_w \in \mathrm{Rnode}(R_h) \wedge PE_w \notin \gamma)$，

(7-2-a) $R_h \notin \mathrm{Cset}(s_i)$ の場合

$$\sum_{w \in \mathrm{Rnode}(R_h) \wedge w \neq z} \beta wz\_R_h \geq 2$$

(7-2-b) $R_h \in \mathrm{Cset}(s_i)$ の場合

$$\sum_{w \neq u \wedge 1 \leq w \leq p} \beta wz\_R_h + \eta uz\_R_h \geq 2$$

(8) もし $\gamma = \emptyset$ かつ，PE_zがすべてのレジスタ R_hを必要としないなら，

$$\gamma uz = 1$$

これは，PE間のメッセージのやりとりが全くなく，しかも責任ノードが替わる場合，PE_uから直接 PE_zへメッセージを送る必要があることを表している．

以上で，フェーズ (6) のγ型メッセージの制約が得られた．フェーズ (5) のψ型メッセージに関しては，γ型メッセージの送信ノードが得られれば機械的に求めることが出来るので，これについては後述する．

メッセージの統合に関して，以下の制約式を与える．

(9) 任意の PE_w, PE_sとレジスタ R_hあるいは，入力変数 x の組について

$$\mu ws \geq \alpha ws, \quad \mu ws \geq \alpha ws\_R_h, \quad \mu ws \geq \alpha ws\_x,$$
$$\mu ws \geq \eta ws\_R_h, \quad \mu ws \geq \eta ws\_x,$$
$$\beta ws \geq \beta ws\_R_h, \quad \beta ws \geq \beta ws\_x$$

各命題変数の決定

上述の (1)~(9) の制約条件を満足するような変数の組の中で，

$$N = \left\{ \sum_{1 \leq y \leq p \wedge y \neq u} \mu uy \right\} + \left\{ \sum_{1 \leq w \leq p \wedge w \neq u} \sum_{1 \leq v \leq p} \beta wv \right\} + \left\{ \sum_{1 \leq y \leq p \wedge y \neq z} \gamma yz \right\}$$

の値を最小にするように各変数の値を定める．

以上で，Nを目的関数する 0-1 整数線形計画問題に帰着できた．最後に，フェーズ (5) のψ型メッセージを定めるために，次のことを行う．

(10) 上述の解で $\gamma wz = 1$ なる PE_wの集合を ξと置く．

(10-1) $|\xi|=1$ のとき $w \in \xi$なる 唯一の wと zについて，$k \neq w \wedge k \neq z$ なる kを，一つ任意に選び，

$$\psi wk = 1, \quad \gamma kz = 1$$

とする．

(10-2) $|\xi| \geq 2$ のとき $w \in \xi$なる 各ノード wに対して，適当な順列を与える．この順列のもとで PE_{p_i}の直後の $PE_{p_{i+1}}$とするとき $\mathrm{next}(p_i) = p_{i+1}$なる関数 next を導入する．順列の最後の要素 p_tと最初の要素 p_0に対して $\mathrm{next}(p_t) = p_0$とする．このとき，各 $PE_w(PE_w \in \xi)$ と PE_k =next(w) に対して，

$$\psi wk = 1$$

$\gamma wz\_R_h = 1$ なら $\psi wk\_R_h = 1, \quad \gamma kz\_R_h = 1$ とする．

5.3.3 各動作仕様の構成方法

ここでは，各 PE_kの導出方法について述べる．一般に，一つのレジスタは2つ以上のノードに分散配置されていると仮定しているので，レジスタ値を送受信するノードの決定には幾つかの自由度がある．これらの自由度の中から通信コストを出来るだけ小さくするようなノードの選択法を **5.3.2節**で述べる．ここでは，それらのノードがすでに決定しているものと仮定して，各 PE_kの導出法を説明する．後で状態遷移図の縮約を行うが，基本的には各 PE_kはもとの全体仕様 SSと同じような形の状態遷移図を持ち，もとの状態遷移図の一つの状態遷移を通信のための動作を含む幾つかの状態遷移の系列に置き換えることにより構成する．

今，全体仕様の状態遷移 t $(= s - \langle a \cdots, G, CR\rangle - s')$ に対して，各 PE_kの状態遷移系列を求めることにする．以下の各フェーズで各 PE_kは指定された状態遷移系列を実行する．各 PE_kごとに，これらの状態遷移系列の順次結合を得ると各 PE_kの状態遷移 t に対する模倣系列が得られる．

説明のための例として，例題の $s_2 \rightarrow s_3$間の状態遷移 $(s_2 - \langle b!r_2, r_3, r_2 > 0, [\cdots]\rangle - s_3)$ を用いる．**図13**は，この遷移に対応する各 PE_kの状態遷移系列である．

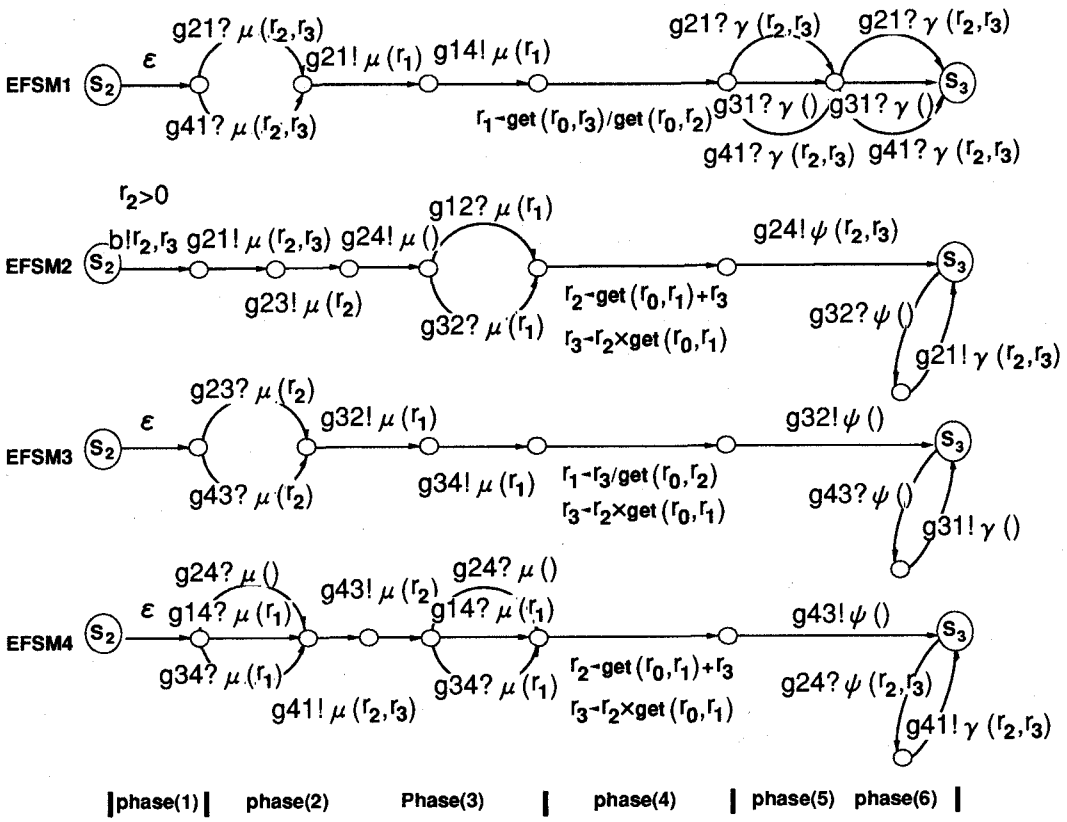


図 13: 各動作仕様 PE_kの遷移 $s_2 - s_3$の状態遷移系列

Fig.13 Implementation of Transition $s_2 - s_3$

(1) 責任ノード (この例では，Snode(s_2)= PE_2) は次の状態遷移 $\langle a\cdots, G, CR'\rangle$ が与えられる．ここで，$a\cdots, G$ は t と同じ，CR'は以下のようなレジスタ更新式である．責任ノードに割り当てられたレジスタのうち，Cset 中のレジスタ値で更新できるレジスタのレジスタ更新式と，更新の対象となったレジスタのうち，今後送信メッセージ中に使うレジスタ値の旧値を r_0に待避させるレジスタ更新式．その他の PEは，ϵ遷移を与える．

(2) 責任ノードは $gki!\mu();\cdots;gkj!\mu()$ が与えられる．ここで，送信動作は送るべきμ型メッセージの数だけ繰り返される．順序は適当でよい．

図 13の例では，$g21!\mu(r_1,r_2)$; $g23!\mu(r_2)$; $g24!\mu()$ となる．ここでは，$\mu(r_1,r_2)$ でレジスタ r_1, r_2の値を持つμ型メッセージを表している．

(3) μ型メッセージの受信，β型メッセージの送信，受信に関する動作は少々複雑である．今，PE_kをμ型メッセージとβ型メッセージを共に受信するようなノードとする．一般にμ型メッセージとβ型メッセージの到着順序は一意ではない．また，このノードが受信する可能性のあるμ型，β型メッセージ数を n とするとき，リンク故障により，$n-1$ 個のメッセージしか受信できない場合もある．これらの点をふまえて，上述のようなノードは，以下のような遷移系列が与えられる．n 個の (μ or β) 型メッセージを $m_1,\ldots,m_n$ とし，これらが $PE_{i_1},\ldots,PE_{i_n}$ から送信されるとする．

$R >> gki!\beta();\cdots;gkj!\beta() >> \underbrace{R >> \cdots >> R}_{n-2}$

where $R = \langle gki_1?m_1, G_1, [\cdots]\rangle[]\cdots[]\langle gki'_n!m_n, G_n, []\rangle$

すなわち，1 つの (μ or β) 型メッセージを受信した段階で，そのノードが送信すべきすべてのβ型メッセージを順次送信する．その後，μ, β型メッセージの受信動作を再び行うが，リンク故障を仮定しているので，残りの $n-2$ 個のメッセージを受信した段階 (全体で $n-1$ 個) で次のフェーズに進むように定める．ここで，G_hは各メッセージの受信条件である (以降では簡単のため省略)．**図 14**(a) はそのための状態遷移系列の構成図である．なお，PE_kがμ型メッセージを受信しないような場合も同様の状態遷移系列で実現する．この場合，β型メッセージの送信部分はない．

次に PE_kが μ型メッセージのみを受信し，β型メッセージを受信しないような場合を考える．この場合，この遷移で PE_kにはただ一つの μ型メッセージしか送信されない．また，PE_kはレジスタ更新を行わず，他のノードのレジスタ更新に必要なレジスタ値の送信 (β型メッセージの送信) のみがその仕事となる．ところで，リンク故障のために，このようなノードにμ型メッセージが到着しないこともありうる．そこで，状態遷移系列を図 **14**(b) のように定める．図 **14**(b) では，μ型メッセージが消失し，受信できない場合を考慮し，このフェーズの状態遷移の開始状態 (フェーズ (1) の終了状態) と終了状態 (フェーズ (4) の開始状態) は同一状態にしている．

図 **13** の例では，すべての PE_kがレジスタ更新を行うので，前者の場合に相当する．このフェーズで各 PE_kへ送られるメッセージ数は順に 2,2,2,3 である．各 PE_kはそれぞれ 1,1,1,2 個のメッセージを受信した段階で次のフェーズに進む．メッセージの到着順は不定であるので，例えば，PE_4では，$g24?\mu()$ [] $g14?\mu(r_1)$ [] $g34?\mu(r_1)$ を 2 つ直列に並べている (それらの間にβ型メッセージの送信系列を挿入している)．これにより，2 つのメッセージの受信が可能である．一般に各メッセージ m の受信の際，そのメッセージにレジスタ値や入力変数の値が入っている．これらの値は後のレジスタ値の更新フェーズ (フェーズ (4)) で用いるので，それらの値をレジスタ r_0に格納する．すなわち，各受信動作でレジスタ更新式 [$r_0 \leftarrow \mathrm{put}(r_0, m)$] を実行する (簡単のため図 **13**では省略している)．

(4) PE_kがレジスタ更新を行うノードであるとき，内部動作 i(とレジスタ更新式) で自ノードのレジスタ更新を与える．ただし，更新式の右辺に現れるレジスタで，自ノードが保持しないレジスタ r'については，その値がレジスタ r_0に保持されているので，その出現を $\mathrm{get}(r_0,r')$ に置き換える．例えば，PE_4では，レジスタ r_1が割り当てられていないので，$[r_2 \leftarrow \mathrm{get}(r_0, r_1) + r_3, r_3 \leftarrow r_2 \times \mathrm{get}(r_0,r_1)]$ というレジスタ更新式が得られる．

(5) PE_kが次の責任ノード (この例では，PE_1) 以外で，かつこの状態遷移でレジスタ更新を行う必要があるノードである場合 (この例では PE_1以外のすべての PE_kが相当する)，図 **12(II)** のように，リング状に ψ型メッセージを送り，ψ型メッセージの受信

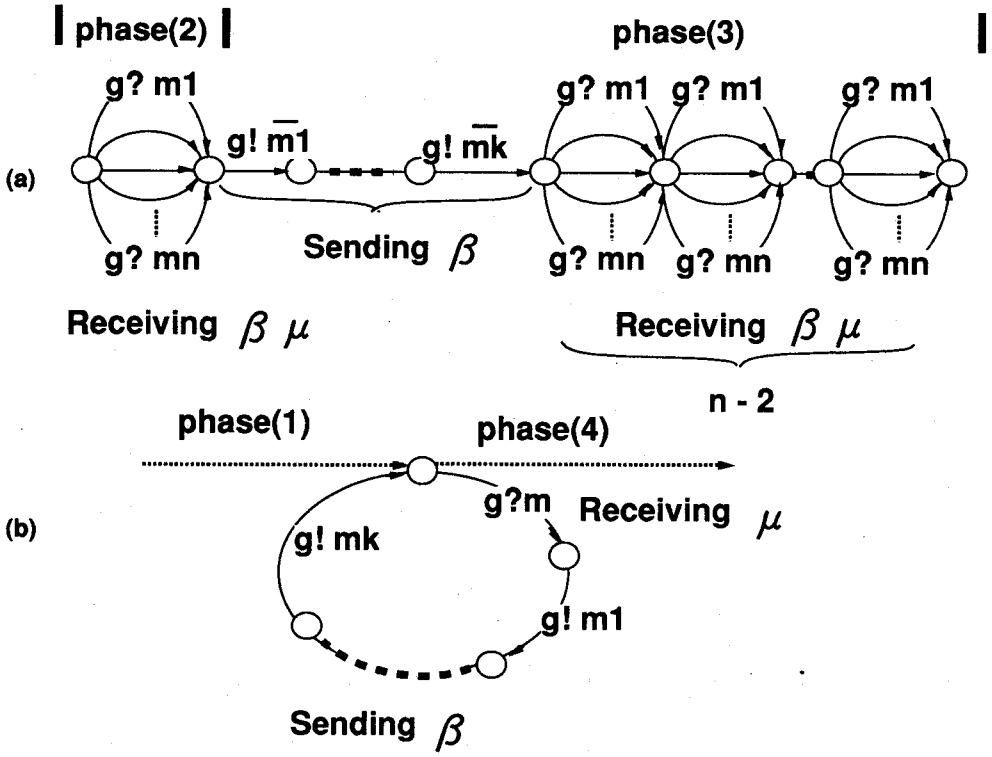


図 14: フェーズ (2)(3) における PE_iの構成

Fig. 14 Phase (2) and (3) in PE_i

に成功したPE_kが次の責任ノード (PE_1) へγ型メッセージを送信するような遷移を得る．3 つのγ型メッセージのうちの 2 つを PE_1が受信すれば，すべてのノードでのレジスタ更新が完了したことが分かる．もし，次責任ノードにメッセージを送る必要のあるノードの数が 1 であれば，普通にメッセージを二重化して送る (図 **12(II)**(a))．

そこで，これらの非責任ノードに相当する状態をすべて一つにまとめϵ遷移を取り除く (**4.3.3節の状態簡約**を参照)．

導出アルゴリズムの全体構成は以下の通りである．

begin

for each transition t^i in SS

$\langle\, ts^i_1, \ldots, ts^i_p \,\rangle$ = make_subsequence(σ, t^i)

$\langle PE_1, \ldots, PE_p \rangle$ = **replace each** t^i **in** $\langle SS, \ldots, SS \rangle$ **to** $\langle ts_1^i, \ldots, ts_p^i \rangle$

$\langle PE_1, \ldots, PE_p \rangle$ = remove_redundant_transitions ($\langle PE_1, \ldots, PE_p \rangle$)

return $\langle PE_1, \ldots, PE_p \rangle$

end

ここで，make_subsequence はリソースの配置情報と全体仕様の一状態遷移を入力として，各動作仕様の模倣系列 $\langle ts_1^i, \ldots, ts_p^i \rangle$ を得る手続きであり，3行目は，全体仕様の各状態遷移を上述の手続きで得られた 模倣系列に置き換える操作を表し，remove_redundant_transitions は状態簡約の手続きである．

5.4 評価

本手法で得られたメッセージの総数は，**図10**の例題の場合 33 メッセージであった．単純に二重化した場合 (45 メッセージ) と比較すると 73% 程度にメッセージ削減できた．

以下のデータで導出時間を計測した．全体仕様 SS は 10 のレジスタを持ち，これらは平均して 5 つのノードに割当てられているとする．各状態遷移で 平均 5 つのレジスタを更新するレジスタ更新式が与えられているとする．このような各状態遷移をランダムに与えたとき，これらの状態遷移に対して，対応する各 PE_k の状態遷移系列を以下の時間で導出した．状態遷移のうち 67% について 20 秒以内，90% について 5 分以内で導出した．なお必ずしも最適解を出すわけではないが，数秒で近似解を求めるグリーディーアルゴリズムを用いた別ルーチンも用意している．

我々の導出方法に関して，以下の定理が成り立つ．

[定理 2] 導出された動作仕様は，仮定している分散環境のもとで，全体仕様と等価である．□

証明 以下が成り立つことを示す．(i) ある動作系列の実行によってノード Snode(S_i) に制御が回ってきた時，次に行うべき動作の実行可能性がそのノードで判定可能であり，かつすべてのリンクは Snode(S_i) がそれまでに受信したタイムスタンプの最大値以下の (古い)

タイムスタンプを持つメッセージしかたまっていない，(ii) その時点までの動作系列の実行によって変化した各レジスタの値は同じ動作系列を全体仕様 SSで実行した時の各レジスタの値に等しい，(iii)Snode(S_i) から始まる一連の動作を実行したとき，高々1ヶ所リンクが故障しても全体として動作が進行し，次の責任ノード Snode(S_j) に制御が渡される．

これらの性質が成り立つことを証明するため，ある状態 S_iの責任ノード Snode(S_i) が一つの動作を選択・実行するとき，上述の性質 (i), (ii) が共に成り立つことを仮定し，一つの動作を実行したとき，上述の性質 (iii) が成り立ち，かつ次の責任ノード Snode(S_j) に制御が移ったとき，性質 (i), (ii) が再び成り立つことを帰納法を用いて証明する．なお，動作の開始時点（初期状態）で性質 (i), (ii) が成り立つことは 3.4 節の制約条件 2 を仮定しているので明らか．

今ノード qを Snode(S_i) とする．ノード qが状態 S_iからの遷移を一つ選んだとき，q以外のノードが選ぶべき遷移はノード qからのメッセージにより一意に定まる．この後，次の (全体仕様 SSに対応する) 状態 S_jに進む際にリンク故障があっても各ノードで必要なレジスタ値が受信できることや，古いタイムスタンプのメッセージに基づいた誤った処理をしないこと，は 4 章の動作仕様 PE^{1-p} の構成法より明らか．よって，性質 (iii) が成り立つ．

これら一連の動作の実行により，次の責任ノード Snode(S_j) に制御が渡される．その際，全体仕様 SSに対応するレジスタ値の更新が行われている．よって，Snode(S_j) に制御が渡された時点でのレジスタ値は性質 (ii) を満たす．また，動作仕様 PE^{1-p}の構成法より Snode(S_j) に制御が渡された時点でノード Snode(S_j) は遷移条件の判定に必要なすべてのレジスタ値を保持しており，自ノードで次に行うべき動作の実行可能性が判定できる．また，一連の送受信動作によってやり取りされるメッセージのタイムスタンプはすべて Snode(S_j) が受信したタイムスタンプの最大値以下である．よって，性質 (ii) が成り立つ． □

提案する模倣方針の妥当性を示す性質として，以下の性質が挙げられる．

[命題 4] 提案する模倣方針は，以下の条件を満たす模倣方針で導出される動作仕様のクラスでは，模倣フェーズ数を最小にする模倣方針の一つである．

条件: 全体仕様の一状態遷移ごとそれを模倣する状態遷移系列を導出すること，リンク故障回避のメッセージの送受信方法として，メッセージの二重化を用いること．(全体仕様の) 各

状態遷移に対して唯一の責任ノードが存在し，このノードがその状態遷移の入出力動作を行うこと．□

上記のクラスに属する模倣方針として，以下の模倣方針も考えられる．その時点での責任ノードが，現時点での全てのレジスタ値を知っているとする．この責任ノードが，(本模倣方針と同様に) 入出力動作を行い，さらに，各レジスタの更新後の値をまとめて計算し，これを必要とするノードに二重化して送信し，次責任ノードに各レジスタの更新後の値を二重化して送信するという集中型の方法である．この方法も，模倣フェーズ数を最小にする模倣方針の一つである．この模倣方針と本論文で述べる模倣方針の違いは，本論文で述べる模倣方針の方が分散性が高いことである．これにより，本模倣方針は，一般に，模倣効率の増加が期待できる．

また，本模倣方針を用いる理由として，以下の点が挙げられる．より実行効率のよい動作仕様を得るために，例えば，どのノードがどのタイミングでレジスタの更新後の値の計算を行い，また実際に更新をするかについて自由度を与えると，模倣フェーズ数やメッセージ数を最小にする解を得るときに考えるべき状態空間が大きくなり，動作仕様の導出が困難になること．

実行効率を考慮に入れた場合，模倣フェーズ数や，使用メッセージ数がコスト基準として大事であるという立場から，本導出法では，内部状態遷移数はコスト基準に入れていない．もちろん，これをコスト基準に入れた導出問題も考えられるであろう．

5.5 結言

本章では，リンクでのメッセージ消失を仮定し，その上で分散システムの全体仕様から各ノードの動作仕様を自動生成する方法について述べた．本方式は，メッセージ決定のための制約式を変えることによって，一般の通信ネットワーク構造や，ノード故障への対応にも発展させることが可能であろう．

本論文で述べた導出方法では，一状態遷移ごとにそれを模倣する動作系列を導出する．複数の状態遷移に着目するとさらに最適化できる可能性がある．これについて，さらなる最適化を考えるのも今後の課題の一つとして考えられる．

6 結論

本研究によって得られた成果, 及び今後の課題をまとめる.

順序機械型モデルに対する階層的設計の際の有用な概念として,「拡張射影」と「到達正当性条件」の二つの概念を導入し, これらの定義, 詳細化の正しさの定義, 及び詳細化の正しさを保証するための証明方法を与えた. 提案する証明法は従来の記法に対するものと比べ多くの作業量を要するものでないことも示した. また, これらの概念を用いて, 在庫管理を最小限の要求のみを指定した要求記述レベルから実行プログラムレベルまで詳細化することができた. この結果より, 従来の詳細化設計法にくらべ, 記述の自由度を高め, しかも, 詳細化の正しさの証明の作業量を増やさないで, 実用プログラムが階層的に記述できることが確かめられた.

「拡張射影」に関して, 詳細化の正しさを保証するための十分条件を関数型プログラムなどの他の記述形態のものに対して考案することや,「拡張射影」を含むテキストの諸性質を調べることは興味深いと思われる. 具体的な証明の実際に関しては文献 [48, 49] などで, 検証者の証明作業の省力化のためのシステム, 記述法, 証明法を研究中である. これらの研究により, 証明者は本質的な証明戦略の考案等に集中できる可能性がある.

また, 順序機械型モデルで記述した全体仕様から, 交換メッセージ数が少ないという意味で効率の良い各ノードの動作仕様を自動導出するアルゴリズムを与えた. またこの導出アルゴリズムが十数個程度のレジスタ数であれば, 0-1 整数線形計画問題の普通の解法を用いても, 実用時間で解を得ることを確認した.

本研究では, メッセージの数量をコスト基準としたが, メッセージサイズ等もコスト基準におくのが望ましい場合も考えられる. この場合の自動導出方法を考案することや, 制御部に並列動作や割り込みを許した全体仕様から動作仕様群を導出することは興味深いと思われる. 現在, 並列動作を含むモデルへの拡張[35, 41, 42]や, 本論文による分散システム設計法のグループワーク支援システム向けの拡張[38, 40, 47]を進めつつある.

謝辞

本研究を行なうにあたり, 常日頃より適切な御指導を賜わり, また, 忍耐強く励ましてくださいました大阪大学基礎工学部 情報工学科 谷口 健一 教授に心から深謝申し上げます.

本研究をまとめるにあたり, 有益な御助言をあたえてくださいました情報工学科の 首藤 勝 教授, 藤井 護 教授, 萩原 兼一 教授に深謝申し上げます.

筆者が, 大阪大学 基礎工学部 情報工学科および同大学院 に在籍中に, 御指導, 御教授くださった 情報工学科の 菊野 亨 教授, 都倉 信樹 教授, 鳥居 宏次 教授, 柏原 敏伸 教授, 宮原 秀夫 教授, 西川 清史 教授, 橋本 昭洋 教授, 奈良先端科学技術大学院大学の 嵩 忠雄 教授, 岡山大学の 杉山 裕二 教授に深謝申し上げます.

本研究をすすめるにあたり, 適切な御助言をしてくださいました本学 情報工学科の 東野 輝夫 助教授, 情報教育センターの 松浦 敏雄 助教授, 情報工学科の 北道 淳司 助手に深謝いたします.

本研究に関して, 有益な御助言をくださいました奈良先端科学技術大学院大学の 関 浩之 助教授, 本学 情報工学科の樋口 昌宏 助手に深謝いたします.

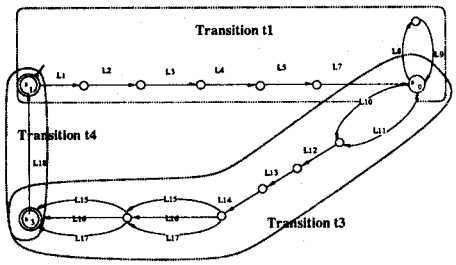
常日頃より御討論していただいた 谷口 研究室の方々に心から感謝いたします.

日本の歴史にきざまれるであろう大災害兵庫県南部地震 (阪神大震災) の最中にこの論文は作成されました. この論文の作成にあたり多大なご尽力をくださいました 谷口 健一 教授, 菊野 亨 教授, 東野 輝夫 助教授, 松浦 敏雄 助教授に改めて深謝いたします. また 飯田 元 助手に深謝いたします.

事務処理を手伝っていただいた事務官の方々に感謝いたします.

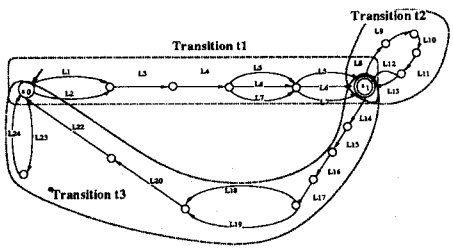
付録

例題から得られた各動作仕様 $PE_k(1 \leq k \leq 4)$ を以下にあげる．PE_kごとの各ラベルの内容の一覧においては，カウンタレジスタに関する操作，遷移条件等は省略している．



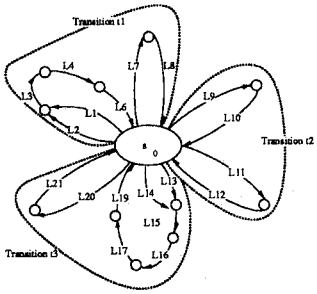
L_1	$a?x,y,z,$	$r_0 \leftarrow \mathrm{put}(r_0,\{x,y,z\})$
L_2	$g12!\mu(x,y,z)$	
L_3	$g13!\mu(x,y,z)$	
L_4	$g14!\mu(x,y,z)$	
L_5	i	$r_1 \leftarrow \mathrm{get}(r_0,x)$
L_7	$g14!\psi$	
L_8	$g31?\psi$	
L_9	$g12?\gamma$	
L_{10}	$g21?\mu(r_2,r_3),$	$r_0 \leftarrow \mathrm{put}(r_0,\mu(r_2,r_3))$
L_{11}	$g41?\beta(r_2,r_3),$	$r_0 \leftarrow \mathrm{put}(r_0,\beta(r_2,r_3))$
L_{12}	$g12!\mu(r_1)$	
L_{13}	$g14!\mu(r_1)$	
L_{14}	i	$r_1 \leftarrow \mathrm{get}(r_0,r_3)$ / $\mathrm{get}(r_0,r_2)$
L_{15}	$g21?\gamma(r_2,r_3),$	$r_0 \leftarrow \mathrm{put}(r_0,\gamma(r_2,r_3))$
L_{16}	$g41?\gamma(r_2,r_3),$	$r_0 \leftarrow \mathrm{put}(r_0,\gamma(r_2,r_3))$
L_{17}	$g31?\gamma$	
L_{18}	$a!\ r_1, \mathrm{get}(r_0,r_2), \mathrm{get}(r_0,r_3)$	

PE_1



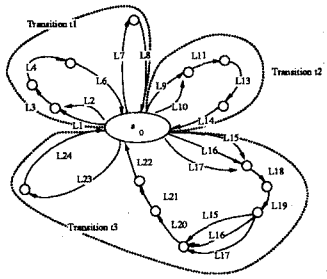
L_1	$g12?\mu(x,y,z),$	$r_0 \leftarrow \mathrm{put}(r_0,\mu(x,y,z))$
L_2	$g32?\beta(x,y,z),$	$r_0 \leftarrow \mathrm{put}(r_0,\beta(x,y,z))$
L_3	$g24!\beta(x,y,z)$	
L_4	i	$r_2 \leftarrow \mathrm{get}(r_0,y),\ r_3 \leftarrow \mathrm{get}(r_0,z)$
L_5	$g12?\gamma$	
L_6	$g42?\gamma$	
L_7	$g32?\gamma$	
L_8	$b?y,$	$r_2 \leftarrow y$
L_9	$g24!\mu(y)$	
L_{10}	$g23!\mu(y)$	
L_{11}	i	$r_2 \leftarrow \mathrm{get}(r_0,y)$
L_{12}	$g42?\gamma$	
L_{13}	$g32?\gamma$	
L_{14}	$r_2 > 0,\ b!r_2,r_3$	
L_{15}	$g21!\mu(r_2,r_3)$	
L_{16}	$g23!\mu(r_2)$	
L_{17}	$g24!\mu$	
L_{18}	$g12?\beta(r_1),$	$r_0 \leftarrow \mathrm{put}(r_0,\beta(r_1))$
L_{19}	$g32?\beta(r_1),$	$r_0 \leftarrow \mathrm{put}(r_0,\beta(r_1))$
L_{20}	i	$r_2 \leftarrow \mathrm{get}(r_0,r_1) + r_3,$
		$r_3 \leftarrow r_2 \times \mathrm{get}(r_0,r_1)$
L_{22}	$g24!\psi(r_2,r_3)$	
L_{23}	$g32?\psi$	
L_{24}	$g21!\gamma$	

PE_2



L_1	$g13?\mu(x,y,z)$,	$r_0 \leftarrow \text{put}(r_0,\mu(x,y,z))$
L_2	$g43?\beta(x,y,z)$,	$r_0 \leftarrow \text{put}(r_0,\beta(x,y,z))$
L_3	$g32!\beta(x,y,z)$	
L_4	i	$r_1 \leftarrow \text{get}(r_0,x)$,
		$r_3 \leftarrow \text{get}(r_0,z)$
L_6	$g31!\psi$	
L_7	$g43?\psi$	
L_8	$g32!\gamma$	
L_9	$g23?\mu(y)$	
L_{10}	$g34!\beta(y)$	
L_{11}	$g43?\psi$	
L_{12}	$g32!\gamma$	
L_{13}	$g23?\mu(r_2)$,	$r_0 \leftarrow \text{put}(r_0,\mu(r_2))$
L_{14}	$g43?\beta(r_2)$,	$r_0 \leftarrow \text{put}(r_0,\beta(r_2))$
L_{15}	$g32!\beta(r_1)$	
L_{16}	$g34!\beta(r_1)$	
L_{17}	i	$r_1 \leftarrow r_3 \,/\, \text{get}(r_0,r_2)$,
		$r_3 \leftarrow \text{get}(r_0,r_2) \times r_1$
L_{19}	$g32!\psi$	
L_{20}	$g43?\psi$	
L_{21}	$g31!\gamma$	

PE_3



L_1	$g14?\mu(x,y,z)$	$r_0 \leftarrow \text{put}(r_0,\mu(x,y,z))$
L_2	$g24?\beta(x,y,z)$	$r_0 \leftarrow \text{put}(r_0,\beta(x,y,z))$
L_3	$g43!\beta(x,y,z)$	
L_4	i	$r_2 \leftarrow \text{get}(r_0,y)$,
		$r_3 \leftarrow \text{get}(r_0,z)$
L_6	$g43!\psi$	
L_7	$g14?\psi$	
L_8	$g42!\gamma$	
L_9	$g24?\mu(y)$,	$r_0 \leftarrow \text{put}(r_0,\mu(y))$
L_{10}	$g34?\beta(y)$,	$r_0 \leftarrow \text{put}(r_0,\beta(y))$
L_{11}	i	$r_2 \leftarrow \text{get}(r_0,y)$
L_{13}	$g42!\gamma$	
L_{14}	$g43!\psi$	
L_{15}	$g24?\mu$	
L_{16}	$g14?\beta(r_1)$,	$r_0 \leftarrow \text{put}(r_0,\beta(r_1))$
L_{17}	$g34?\beta(r_1)$,	$r_0 \leftarrow \text{put}(r_0,\beta(r_1))$
L_{18}	$g41!\beta(r_2)$	
L_{19}	$g43!\beta(r_2)$	
L_{20}	i	$r_2 \leftarrow \text{get}(r_0,\, r_1) + r_3$,
		$r_3 \leftarrow r_2 \times \text{get}(r_0,r_1)$
L_{22}	$g43!\psi$	
L_{23}	$g24?\psi(r_2,r_3)$	
L_{24}	$g41!\gamma(r_2,r_3)$	

PE_4

参考文献

[1] M. Wirsing: Chap. 13, Algebraic Specification, in "Handbook of Theoretical Computer Science Volume B: FORMAL MODEL AND SEMANTICS", (Eds.) J. v. Leeuvwn, Elseveir Science Publishers, (1990).

[2] M. Wirsing, P. Pepper, H. Partsch, W. Dosch and M. Broy: "On hirarchies of Abstract Data Types", Acta Inform., 20, pp.1-33 (1983).

[3] K.Futatsugi, J. Gouguen, J. -P. Jounnaud, and J. Meseguer: "Principles of OBJ2", Symposium of Principles of Programming Languages, pp.52-66 ACM, (1985).

[4] J.V. Guttag, J.J. Horning and J.M. Wing: "Larch in five easy pieces", Tech. Report, Digital Systems Research Center, (1985).

[5] S.J. Garland, and J.V. Guttag: "An Overview of LP, The Larch Prover", **LNCS 355**, pp.137-151, (1989).

[6] J.A Bergstra, J. Heering and P. Klint: "Algebraic Specification", ACM Press, (1989).

[7] 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 (1)—多ソート代数と等式論理", 情報処理,**25**,1, pp. 47-53, (1984), 他に, 情報処理,**25**,5, pp. 491-501, (1984), 情報処理,**25**,7, pp. 708-716, (1984), 情報処理,**25**,9, pp. 971-986, (1984).

[8] B. Potter, J. Sinclair and D. Till: "An Introduction to Formal Specificaiton and Z", International Series in Computer, Science, Prentice-Hall, (1991).

[9] M.J.Gordon: "Mechanizing Programming Logic in Higher Order Logic", in "Current Trends in Hardware Verification and Automated Theorem Proving", (Eds.) G. Birtwistle and P.A. Subrahmayan, pp.387-439, Springer-Verlag, (1989).

[10] 高原 厚: "プロセス代数を用いた形式的検証", 情報処理 **35**, 8, pp.736-741, (1994).

[11] 木村 晋二: “形式的タイミング検証について”, 情報処理 **35**, 8, pp.726-735, (1994).

[12] 二村 良彦, 雨宮 真人, 山崎 利治, 淵 一博: “新しいプログラミングパラダイムによる共通問題の設計”–他, 情報処理,**26**, 5, pp.458-520, (1985).

[13] 山崎 利治:“共通問題によるプログラム設計技法解説”– 他, 情報処理,**25**, 9, pp.934-962, (1984).

[14] 山崎 利治:“共通問題によるプログラム設計技法解説 (その 2)”–他, 情報処理,**25**, 11, pp.1219-1268, (1984).

[15] 湯浅 太一:“モジュラ・プログラム言語イオタによる在庫管理プログラムの仕様記述とプログラム記述”, 情報処理,**26**, 5, pp.486-496, (1985).

[16] 大蘆 雅弘, 杉山 裕二, 谷口 健一:“代数的言語 ASL における抽象的順序機械型プログラムとその処理系”, 電子情報通信学会論文誌 (D1), **J73-D-I**, 12, pp.971-978, (1990).

[17] 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–”, 電子情報通信学会論文誌 (D), **J69-D**, 7, pp.1066-1075, (1986).

[18] 東野 輝夫, 関 浩之, 谷口 健一:“代数的仕様から関数型プログラムの導出とその実行”, 情報処理, **29**, 8, pp.881-896, (1988).

[19] 関 浩之, 井上 克朗, 谷口 健一, 嵩 忠雄:“関数型言語 ASL/F のコンパイル時における最適化”, 電子情報通信学会論文誌 (D), **J67-D**, 10, pp.1115-1122, (1984).

[20] R. D. Gumb:“Programming Logics–An Introduction to Verification and Semantics–”, John Wiley and Sons, pp.90-127 (1988).

[21] R. Probert and K. Saleh : “Synthesis of Communication Protocols: Survey and Assessment”, IEEE Trans. on Comp., Vol.40, No.4, pp.468-476, (1991).

[22] F. Khendek, G. v. Bochmann and C. Kant : “New results on deriving protocol specifications from service specifications”, Proc. of ACM SIGCOMM ’89, pp.136-145, (1989).

[23] G. v. Bochmann and R. Gotzhein : "Deriving protocol specifications from service specifications", Proc. of ACM SIGCOMM '86, pp.148-156, (1986).

[24] R. Langerak : "Decomposition of functionality; a correctness-preserving LOTOS transformation", Proc. of Tenth IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification, North Holland, pp. 229-242, (1990).

[25] T. Higashino : "Service Specification and Its Protocol Specifications in LOTOS", IEICE Trans. Fundamentals, Vol. E75-A, No. 3, pp.330-338, (1992).

[26] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807, (1989).

[27] P.-Y.M. Chu and M.T. Liu : "Synthesizing Protocol Specifications from Service Specifications in FSM model", Proc. Computer Networking Symp. '88, pp.173-182, (1988).

[28] P.-Y.M. Chu and M.T. Liu : "Protocol Synthesis in a State-Transition Model", Proc. COMPSAC'88, pp.505-512, (1988).

[29] D. Park: "Concurrency and automata on infinite sequences", Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 104. pp.167-183, Springer-Verlay, (1981).

[30] R. Milner: "Comunication and Concurrency", Prentice-Hall, (1989).

[31] M. R. Garey, D. S. Johnson: "Computers and Intractability", FreeMan, (1979).

[32] C.V. Ramamorthy, S.T. Dong and Y. Usuda : "Synthesis and performance of two-party error recoverable protocols", Proc. COMPSAC,86, pp.214-220, (1986).

[33] Z. Wang: "Shortest Path with Emergency Exits", Sigcomm '90 Symp. Communications Architectures & Protocol ,pp. 166-pp.176, (1990).

[34] 郷 健太郎, 白鳥 則郎: "等価性に基づく LOTOS 仕様の記述スタイル変換法" 情報処理学会論文誌, Vol. 34, No.6, pp.1281-1289, (1993).

[35] Hirozumi YAMAGUCHI, Kozo OKANO, Teruo HIGASHINO and Kenichi TANIGUCHI: "Synthesis of Protocol Specifications from Service Specifications of Distributed Systems in a Marked Graph Model" IEICE Trans. EA, Vol E77-A, No. 10, pp.1623-1633, (1994).

[36] 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: "代数的言語 ASL で記述した在庫管理プログラムとその正しさの証明", 電子情報通信学会技術報告, SS 90-29, pp. 89-98, (1990).

[37] 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: "拡張有限状態機械モデルを用いた分散処理システムの全体記述から各ノードの動作記述の自動生成", 情報処理学会研究報告, 92–PRG–8, 27, pp. 211-218, (1992).

[38] 今城 広志, 岡野 浩三, 東野 輝夫, 谷口 健一: "拡張有限状態機械を用いた協調作業向きの計算システム", 情報処理学会研究報告, 93–DPS–61,93–OS–60, 20 , pp.147-154, (1993).

[39] 岡野 浩三, 今城 広志, 東野 輝夫, 谷口 健一: "リンクの故障を考慮に入れた分散システムの動作仕様の自動導出 ", 信学会情報基礎論ワークショップ LA シンポジウム報告, 93–LA, 68, pp.70-75, (1993).

[40] 今城 広志, 尹 達雄, 岡野 浩三, 東野 輝夫, 谷口 健一: "協調計算システムの動作仕様群の分散実行系", 情報処理学会研究報告, 94–GW–6, 4, pp.19-24, (1994).

[41] 岡野 浩三, 山口 弘純, 東野 輝夫, 谷口 健一: "レジスタを持つペトリネットで記述された分散システムの要求仕様から各ノードの動作仕様の導出", 情報処理学会研究報告, 94–OS–64, 94–DPS–65, 27, pp.157-162, (1994).

[42] H. YAMAGUCHI, K. OKANO, T. HIGASHINO and K. TANIGUCHI: "Software Process Description in a Petri Net Model and its Distributed Execution", 電子情報

通信学会技術報告, 94–SS–38, pp.25-32, (1994).

[43] 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “代数的言語 ASL を用いたプログラムの設計開発”, 平成 2 年 秋季 電子情報通信学会全国大会, SD-3-1, pp. 6·471-6·472, (1990).

[44] 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “ASL システムを用いたプログラムの全正当性の証明–在庫管理プログラムを例として–”, 平成 3 年 春季 電子情報通信学会全国大会, SD-1-6, pp. 6·301-6·302, (1991).

[45] 岡野 浩三, 東野 輝夫, 谷口 健一: “代数的言語 ASL を用いた酒屋在庫管理の要求仕様記述”, 平成 4 年 春季 情報処理学会全国大会, 4K-2, pp.5·327-5·328, (1992).

[46] 岡野 浩三, 東野 輝夫, 谷口 健一: “ASL を用いた階層的設計における詳細化の正しさとその検証”, 平成 4 年 秋季 電子情報通信学会全国大会, SD-12-1, pp.6·418-6·419, (1992).

[47] 今城 広志, 岡野 浩三, 東野 輝夫, 谷口 健一: “拡張有限状態機械で記述された協調計算プログラムとその実行系”, 平成 5 年 秋季 情報処理学会全国大会, 2E-1, pp. 1·151-1·152, (1993).

[48] 森岡 澄夫, 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “ASL プログラム開発システムにおける検証の自動化について”, 平成 6 年 春季 情報処理学会全国大会, 1G-05, pp. 4·279-4·280, (1994).

[49] 森岡 澄夫, 岡野 浩三, 北道 淳司, 東野 輝夫, 谷口 健一: “整数上の論理式の恒真性判定手続きを利用した順序機械型仕様記述の詳細化の正しさの半自動証明”, 平成 6 年 ソフトウェア科学会全国大会, pp. 361-364 (1994).

表 目 次

図目次