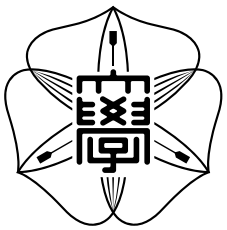
博 士 論 文

宣言的仕様に対する正当なデジタル回路および協調計算システムの自動生成に関する研究



2011 年 3 月

北海道大学 大学院情報科学研究科
複合情報学専攻

吉 川　浩
Hiroshi Yoshikawa

目次

図目次

第 1 章

序論

本研究は仕様からデジタル回路や協調計算システムを自動生成する**高位設計**の研究である．本研究では「仕様記述」と「生成された物」の間の正当性が数学的に保証される枠組みを重視している．このため，仕様と生成物の間の正当性が保証される**等価変換理論** (*Equivalent Transformation Theory : ET*) [1–9] の枠組みを用いた．

本研究では仕様の記述方法として**宣言的仕様**を用いる．宣言的仕様とは，問題の性質と解く範囲を**確定節**によって定義したものであり，一般的な意味の「仕様」より狭い意味で用いられる．宣言的仕様が扱う問題には**求解問題**や**制約充足問題**などがあり，本研究の目的は，このような問題を効率よく解くためのデジタル回路やデジタル回路とソフトウエアによる協調計算システムを自動生成することである．

1.1 概要

本書の中で，宣言的仕様から正しいデジタル回路を自動生成する枠組みを提案する．この枠組みの中では，複雑な問題も宣言的仕様によって容易に記述できる．回路は，その記述から数学的な基礎に基づいて正しく自動生成される．従って，開発の難しさが解消され，安全に短期開発が可能となる．

問題を解くアルゴリズムやプロシージャは宣言的仕様に含まれない．これらは**等価変換理論** (ET 理論) に基づいて自動生成され，**ET ルール**と呼ばれる確定節に対する書き換え規則によって記述される．本研究が提案する枠組みの特徴は，宣言的仕様から ET ルールを生成し，それをプログラム変換によって回路レベルの記述へ変換していることである．5 章では，この変換過程を理論的に考察している．この中で議論される**代数系変換**の理論は，計算のドメインを変更するようなプログラム変換に対しても正当性を与える．

代数系変換の理論は ET ルールから回路を生成する場合だけでなく，ET ルールからソフトウエアを生成する場合にも使える．そこで，回路とソフトの**協調計算システム**を生成する枠組みを 6 章で提案する．

本研究では協調計算を容易に実験するため，柔軟な協調計算実験システムを制作している．7 章では，このシステムの概要と「お絵かきロジック」を用いたデモを紹介する．

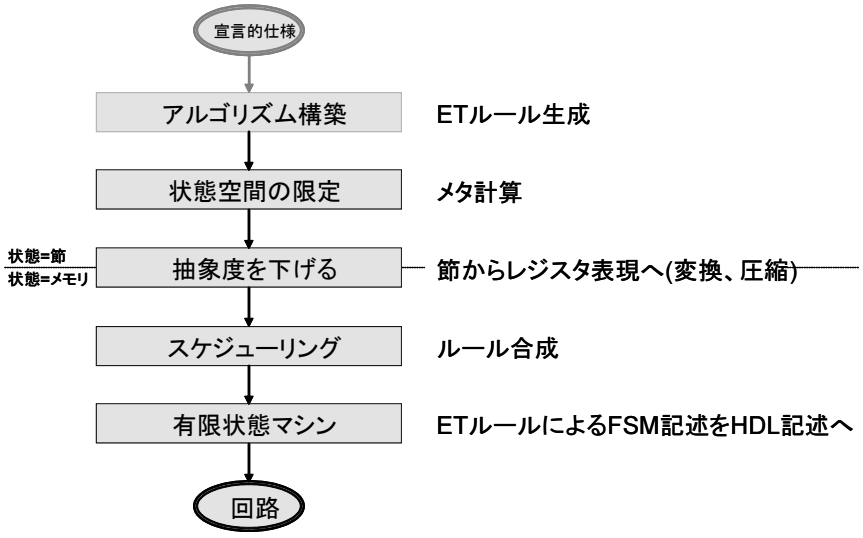


図 1.1　デジタル回路を生成する流れ

1.1.1　宣言的仕様からデジタル回路を生成する流れ

宣言的仕様から回路を生成する流れを概説する (図 1.1).

1. **アルゴリズム構築**　まず，宣言的仕様から **ET ルール**と呼ばれる書き換え規則を生成する必要がある．これは宣言的仕様からアルゴリズムを生成することに相当する．
   この ET ルール生成に関する研究は数多くなされており [7–12]，本研究ではそれらの研究成果を用いて「ET ルールが生成された」と仮定して議論を進める．本研究が新たにおこなったのは「与えられた ET ルールからデジタル回路を生成する」部分，およびフレームワークの提案である．
2. **状態空間の限定**　状態空間とは計算過程に現れる全ての状態の集合である．デジタル回路で実現可能なものは有限状態マシン [13–15] であるため，回路化可能な ET ルールの状態空間も有限である．この空間を**メタ計算**によって明らかにする．
3. **抽象度を下げる**　ET ルールは確定節と呼ばれる論理式を書き換える規則である．確定節は宣言的仕様にも使われており，表現の抽象度が高い．この確定節による表現をレジスタを用いた表現へ変換する．これには**プログラム変換**という手法を用いる．このフェーズでは，必要なレジスタ数を圧縮する操作も行う．
4. **スケジューリング**　上記のプログラム変換されたルールからデジタル回路を生成するには，有限状態マシンをあらわすルールを生成する必要がある．このルールは上記のプログラム変換されたルールをマージすることで実現される．最終的にルールは一つになる．

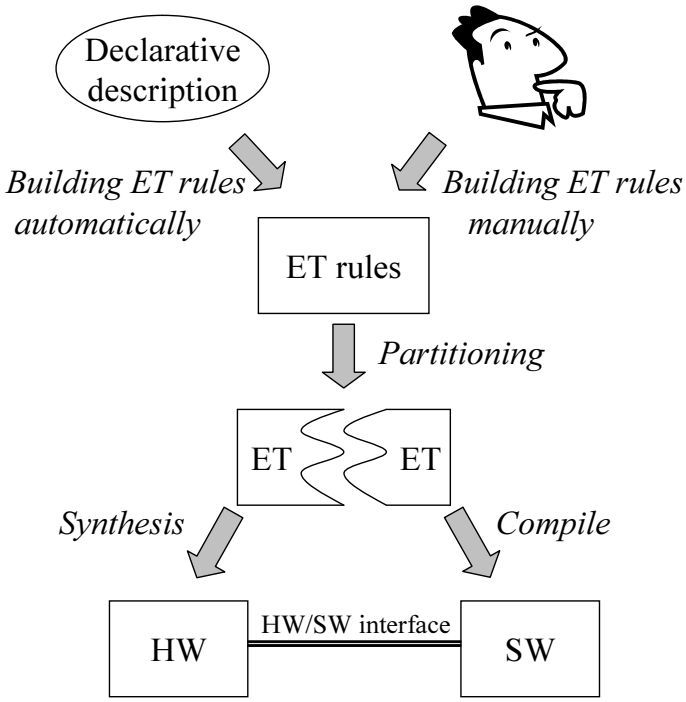


図 1.2　Co–design framework using ET

5. **有限状態マシン**　マージされて得られたルールの形は，有限状態マシンとの対応が自明になっている．したがって，ルールのシンタックスから HDL(VHDL[16], Verilog HDL[17]) のシンタックスへ変換すればよい．

1.1.2　宣言的仕様から協調計算システムを生成する流れ

回路生成の方法を応用することで，宣言的仕様から協調計算回路を生成できる．その流れを概説する (図 1.2).

1. **ET ルール生成**　ET ルールを生成する．宣言的仕様から自動生成することを考えているが，人がマニュアルで書いても構わない．また，デジタル回路の生成の場合とは異なり回路化できない記述も許される．
2. **機能分割**　ET ルールを「デジタル回路化するルール」と「ソフトウェア化するルール」に分類する．ET ルールは書き換え規則であり，書き換え規則は個々の規則が独立しているため機能分割が容易である．これは手続き型言語 (C/C++, Java, etc.) にはない特徴である．
3. **協調計算システムの生成**「デジタル回路化するルール」を本研究の手法を用いてデジタル回路へ変換し,「ソフトウェア化するルール」を関連研究 [12, 18–20] を用いてコンピュータ上で実行可能なソフトウェアへ変換する．

1.2 構成

本書の構成は以下のとおりである．

1 章 序論
この研究の要旨と背景について述べる．またデジタル回路および協調計算システムについての説明も行う．

2 章 宣言的仕様，宣言的意味と正当性
この研究で用いる宣言的仕様について，最初に記述言語として用いる確定節について説明し，続いて宣言的意味と正当性について説明する．

3 章 等価変換と ET ルール
本研究がハードウェア記述言語 (HDL) として用いている *ET* **ルール**の説明と，その ET ルールに対して正当性を保証する理論である**等価変換理論**について説明する．

4 章 ET ルールを用いた回路生成手法
ET ルールの記述からどのように回路が生成できるのか，その基礎的なアイデアを説明する．

5 章 回路生成手法の理論的考察と拡張
4 章のアイデアを理論的に考察し，回路の正当性について論じる．また，その成果として回路生成方法を拡張する理論を展開する．

6 章 ET ルール変換フレームワーク
5 章で得られた知見をもとに，一般的な ET ルールの**プログラム変換**のフレームワークを示す．このフレームワークにより，回路だけでなくコンピュータ上で実行可能なソフトウェアの生成も可能になる．そして，回路とソフトの協調計算システム生成へも展開可能になる．

7 章 協調計算システムへの応用
実際に実験を行うため，柔軟な協調計算システムを試作した例を紹介する．この中で「お絵かきロジック (Nonogram)」と呼ばれるパズルを用いて協調計算の効果を示している．

8 章 結論

1.3 デジタル回路と協調計算システム

ここでは本書が扱うデジタル回路と協調計算システムについての基本的な概念について説明する．その目的は，デジタル回路設計の制約の厳しさを理解することにある．

最初にデジタル回路について説明する．その中で，デジタル回路の設計では**ポインタやスタックなどが利用できない**ことを説明する．この事実は，宣言的仕様の中にはデジタル回路だけでは実現不可能な場合があることを意味する．そのため，ソフトウェアとの協調計算も重要になる．この章の最後では，本研究が生成しようとする協調計算システムにつ

いても説明する．

1.3.1 デジタル回路の概要

ここではデジタル回路の基本的な知識を説明する．まず，デジタル回路を構成する基本単位である**ゲート**と**フリップフロップ**について説明し，それらがどのように組み合わされて論理的な機能が実現されるのかを説明する．この中で**除算**を組み合わせ回路で実現することが困難であることや，デジタル回路にはポインタやスタックが利用できないことも説明する．

1.3.2 本書が対象とする範囲

デジタル IC のパッケージの内部にはシリコン基板上に形成された論理回路があり，そのシリコン基板上の論理回路からワイヤーボンディングを介して回路の入出力が IC のピンに接続されている．

本書が扱うのは内部の論理回路の**「論理」**の設計である．すなわち，後述の**ゲート**と**フリップフロップ**で表現されるレベルだけを扱い，ワイヤーボンディングや IC のピンなどのパッケージや電気的な物理特性などは扱っていない．

1.3.3 デジタル回路の基本素子

デジタル回路は 2 種類のトランジスタ (n–MOS および p–MOS) をシリコン基板上に集積したものである．トランジスタは高速にオン・オフする電気的なスイッチ素子であり，構造上の最小構成要素である．しかし一般にデジタル回路の設計では，いくつかのトランジスタを組み合わせて作られる**最小の論理機能**を持った回路を基本素子と考える．その最小の論理機能とは次の 2 種類である．

1. ゲート
2. フリップフロップ

これらの基本素子の内部構造はメーカーや IC の種類によって異なるが，機能はメーカーや IC の種類にかかわらず同等である．デジタル回路は，この 2 種類の基本素子を組み合わせて実現されている．以下に，それぞれの基本素子の詳細を説明する．

■ゲート (gate circuit)　ゲートは基本的なブール演算を行う回路である．電圧の高低 (High / Low) によってブール代数の値 (1 / 0) をあらわし，High 電圧の状態を “1”，Low 電圧の状態を “0” に割り当て[*1]，入力電圧の状態によって出力電圧を決定する．

[*1] この表現方法を**正論理** (*Positive Logic*) と呼ぶ．逆の場合 (High = 0, Low = 1) は**負論理** (*Negative Logic*) と呼ばれる．本書は正論理で説明する．

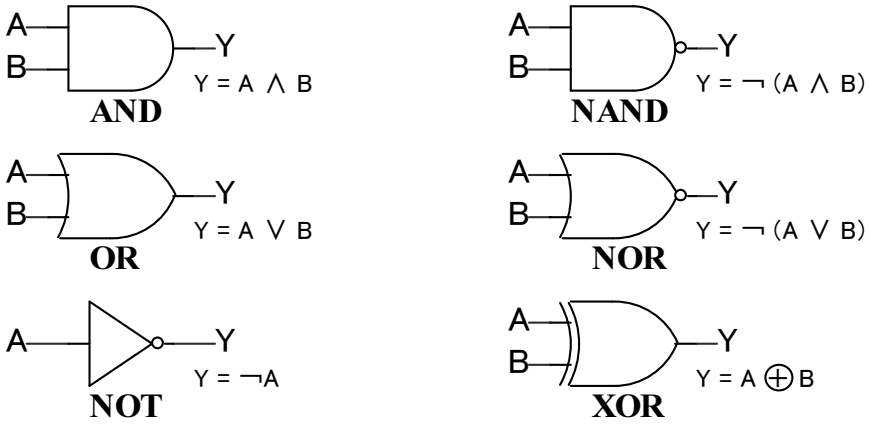


図 1.3　ゲートの回路記号

- **ゲートの種類**　ゲートにはいくつかの種類がある．図 1.3 は代表的なゲートを回路図記号と共に示している．記号の “A” および “B” はゲート回路の入力信号で，“Y” はゲート回路の出力信号 (つまり演算結果) をあらわしている．その下の論理式は出力 Y と入力 A,B の関係を示している．
  回路設計には最低限 **AND**, **OR**, **NOT** の 3 つが必要である．しかし多くの設計ツール (EDA Tool) には，**NAND** (Not–AND), **NOR** (Not–OR), **XOR** (eXclusive–OR 排他的論理和) なども用意されている．
- **ゲートの用途**　ゲートは関数を実現する目的に使用される．理論上は AND, OR, NOT の三種類の演算機能を組み合わせて任意の関数を作れることが知られている．ただし関数によっては必要なゲート数が莫大になるものがあり (例えば除算回路), このような関数は後述の**順序回路**で実現される．
- **ゲート回路の演算時間**　一般にゲート回路は入力信号に対する演算結果がほぼ瞬時に得られる．ゲート回路が演算に要する時間 (すなわちゲート回路に信号を入力してからゲート回路の出力が確定するまでの時間) を**伝搬遅延** (*propagation delay*) と呼ぶ (図 1.4 の δ)．伝搬遅延とは，物理的な制約で生じる**電気信号の遅れ**である．制約とは，配線を電気が伝わる速さが有限であることや，トランジスタのオン・オフに要する時間をゼロにできないことなどである．
  伝搬遅延の大きさを決める要因 (パラメータ) には，動作電圧, 回路の構造や寸法, シリコンの特性など様々なものがある．このため伝搬遅延を厳密にコントロールすることは難しいが，一般に動作クロックの周期に比べて十分に小さいので無視することができる[*2]．

[*2] むしろ，伝搬遅延を無視できるように動作クロックの周波数範囲に制約が課される．

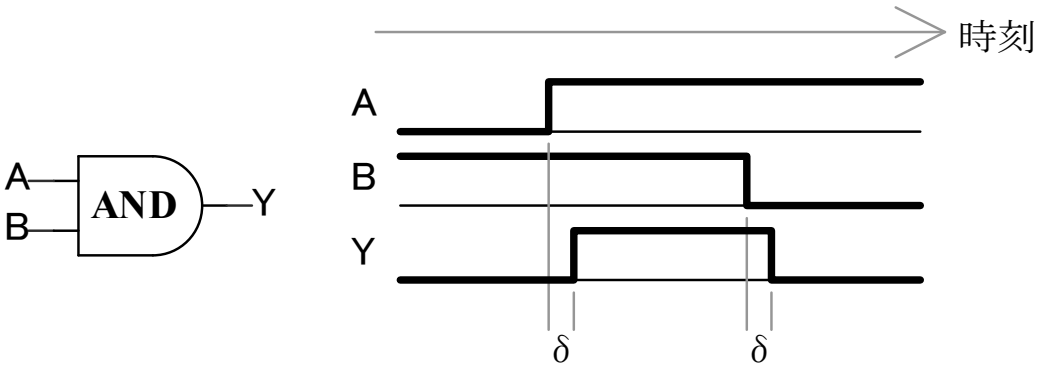


図 1.4　ゲートの伝搬遅延

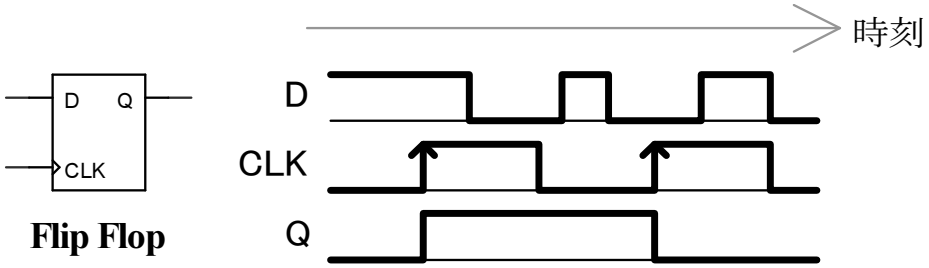


図 1.5　フリップフロップ

本研究では**ゲート回路の伝搬遅延は無視する**ものとし，ゲート回路の計算時間を理想的にゼロとみなす．

■フリップフロップ (Flip–Flop, F/F)　フリップフロップ (F/F) は最も単純な記憶素子であり，1 個の F/F は 1 ビットの状態を保持する．多ビットの状態を保持するには複数の F/F を用いる．図 1.5 は F/F の回路図記号と動作を示している．F/F は入力 (D) と出力 (Q) のほかにクロック入力 (CLK) を持つ．F/F は回路の配線の途中に挿入され，クロック信号 CLK が Low から High へ変化するタイミングで入力 D を読み取り，それを Q へ出力する．これ以外のタイミングでは出力の状態が保持され，その間の入力信号の変化は無視される．

F/F は一種のメモリではあるが**アドレス**という概念を持たない．したがって，デジタル回路の設計では**ポインタやスタックなどが利用できない**ことに留意しておく必要がある[*3]．

[*3] 基本素子（ゲート回路と F/F）を組み合わせてポインタやスタックの機能を実現することは可能である．しかし，この機能を持つ基本素子が存在するわけではない．

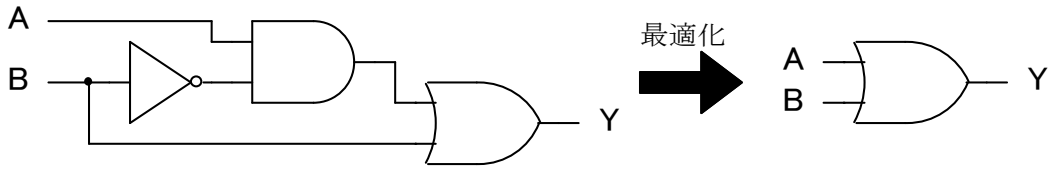


図 1.6　組み合わせ回路と最適化の例

1.3.4　基本素子で実現されるデジタル回路

実用的なデジタル回路は基本単位であるゲートとフリップフロップが複雑に組み合わされたものである．しかし，その複雑な回路も次の二種類の基本的な構成方法を用いて組み立てられている．

- **組み合わせ回路** (Combinational Logic Circuits)
- **順序回路** (Sequential Circuits)

どのような複雑なデジタル回路も，これらの構成方法が基本となっている．ここでは，組み合わせ回路および順序回路と，順序回路の特別な形である**有限状態マシン** (*Finite State Machine : FSM*) について説明する．

■組み合わせ回路 (Combinational Logic Circuits)　デジタル回路の基本素子のうち，ゲート回路の組み合わせだけで構成される回路を**組み合わせ回路**と呼ぶ．ゲート回路は理想状態では計算時間をゼロとみなせるため，組み合わせ回路の計算時間も理想状態ではゼロとみなせる．

組み合わせ回路で実現できるものは関数である．任意の関数は選言標準形の論理式であらわすことができるので原理的には AND，OR，NOT の三種類のゲート回路だけで任意の関数を実現できる．多くの CAD では加減算, 乗算, 比較などを使った数式を自動的に組み合わせ回路へ変換し最適化する機能を備えている．したがって，人の手によって組み合わせ回路を設計することは少ない．**本研究でも組み合わせ回路の合成と最適化はツールに任せ，数式レベルの記述を生成する**．

- **組み合わせ回路の最適化**　通常，組み合わせ回路は，実装時に回路が簡単になるように論理の最適化を行う (図 1.6)．有名な最適化手法としてはカルノー図 (Karnaugh map) による最適化 [21] や，コンピュータでの自動化に適したクワイン・マクラスキー法 (QM 法)[22, 23] がある．QM 法は，1 ビット出力の論理式に対する最適化手法であり，元の論理式と等価で項数が最少の論理式が得られることを保証する．しかし，デジタル回路にとって項数が最少であることは必ずしも最適ではなく，その時代の技術 (**テクノロジ・マッピング**) などによっても最適解は異なる．また，QM 法は入力ビット数の増加に対して計算量が爆発的に増えるため，実用レベル

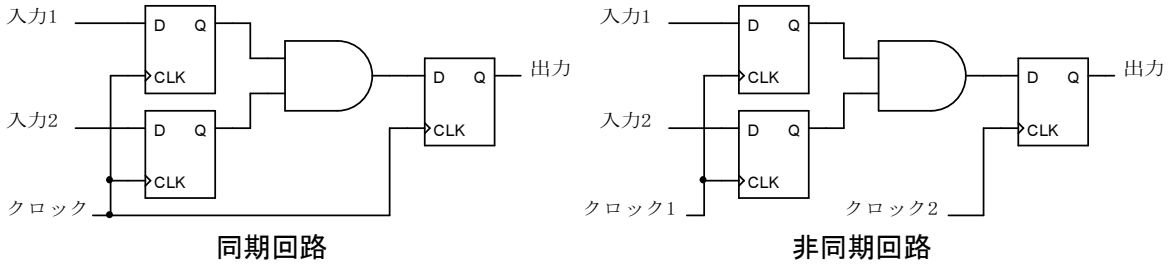


図 1.7　同期回路と非同期回路の例

ではヒューリスティックを用いた最適化法 [24, 25] や二分決定図 (Binary Decision Diagram : BDD) を用いた最適化法が用いられている．
加減算や乗算に対しては上記の最適化は有効に働かない．しかし，これらの回路に対しては演算方法の特徴を巧みに利用した個別の手法が考案されており，組み合わせ回路で実現可能となっている．

- **加減算回路**　最も単純な Ripple Carry Adder (RCA) や，実用的な Carry Saved Adder (CSA) などが考案されている．
- **乗算回路**　Wallace Tree[26] による乗算やブース符号 [27] による乗算などが考案されている．

- **組み合わせ回路で実現が難しい関数**　上記の最適化法を用いても回路規模が実用的な規模に収まらないことは多い．このような回路の代表的なものに**除算回路**がある．除算回路は最適化しても実用的な回路サイズにならず，加減算や乗算のような巧みな手法も発見されていない．このように有効な最適化ができない場合には，後述の順序回路を用いてアルゴリズム的に繰り返し計算するのが一般的である．

■順序回路 (Sequential Circuits)　組み合わせ回路とフリップフロップを用いて構成される回路である．大きく同期回路と非同期回路に分けられる (図 1.7).

- **同期回路 (Synchronous Circuits)**
  順序回路のうち，全てのフリップフロップのクロック入力が同じクロック信号によって駆動されているものを同期回路と呼ぶ．クロック信号が変化しない間は状態を保持し，その状態と外部入力に応じて次状態を計算し，クロック信号の変化に同期して次状態に遷移する．
- **非同期回路 (Asynchronous Circuits)**
  異なるクロック信号で駆動されるフリップフロップを持つ順序回路を非同期回路という．クロック信号の役割は，フリップフロップがデータを取り込むタイミングを合図することである．このため，異なるクロック信号で動作するフリップフロップの間では**データを受け渡すタイミングの調整が難しく誤動作しやすくなる**．

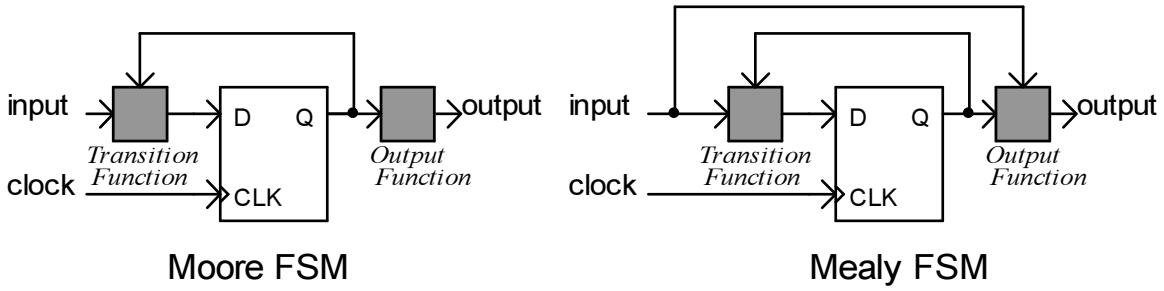


図 1.8　ムーア型状態マシンとミーリ型状態マシン

デジタル回路の設計では非同期回路を避け同期回路を用いた設計が推奨されている．本研究で生成されるデジタル回路も同期回路である．

■有限状態マシン (Finite State Machine : FSM)　順序回路の特別なものとして有限状態マシン (Finite State Machine : FSM) [13–15] がある．FSM はフリップフロップの内容を**内部状態** (*State*) と捉え，次に遷移すべき状態を現在の内部状態と外部入力から**遷移関数** (*Transition Function*) によって決定する．一般的な FSM は**出力関数** (*Output Function*) によって出力を決定する．出力が内部状態のみで決定されるものを**ムーア型** [14]，内部状態と外部入力から決定されるものを**ミーリ型** [15] と呼ぶ (図 1.8)．本研究はムーア型 FSM を用いて宣言的仕様からデジタル回路を生成している．

1.3.5　デジタル回路の特徴

個々のデジタル回路は処理手順が固定化 (*hard–wired*) されており，CPU のようにメモリから命令を読み込む手間がかからない．また，デジタル回路では記憶素子であるフリップフロップ (F/F) が処理回路の内部に埋め込まれている．このため**ノイマンボトルネックは存在しない**．また CPU の動作は逐次処理しかできないが，デジタル回路は複数の回路を並列に実行することが可能である．デジタル回路を利用する多くのアプリケーションでは，並列実行可能性を利用して高効率な計算を実現している．

その代わり F/F には**アドレスの概念がない**ため，F/F を使って処理手順を変更したり，スタックやポインタを利用した処理ができない．この制約のために，デジタル回路で実現可能な処理には限界がある．

配線上のデータの流れ

ゲートとフリップフロップは入力と出力が明確に決まっており，**データの流れは一方向**である．**出力**とは，そこに接続される配線の電圧を High または Low の状態に強制的に駆動するものである．**入力**とは，そこに接続されている配線の電圧に影響を与えずに電圧の状態を読み取るものである．

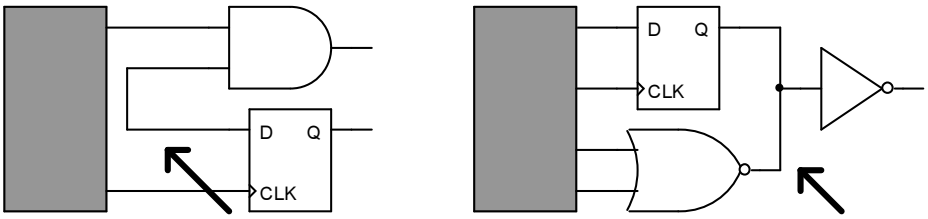


図 1.9　許されない配線方法の例

デジタル回路を設計する上では次のことが非常に重要である．

- ひとつの配線には，ただ一つの出力と 0 個以上の入力が接続されなければならない．

特に出力は必ず一つだけ接続されなければならない (図 1.9)．もし出力がひとつも接続されない配線が存在すると，その配線の電圧は定まらない (つまり真理値が決まらない)．また複数の出力が接続された場合，それらの中に矛盾した出力があるとき (つまり High と Low が混在するとき)，電圧が定まらないばかりか High から Low へ向かって大きなショート電流が流れて回路の発熱や発火事故が発生する可能性がある．

1.3.6　デジタル IC の種類

デジタル IC は，構造によって大きく 4 つに分類される．

1. フルカスタム
2. Gate Array (ASIC)
3. PLD / CPLD
4. FPGA

それぞれについて下記に説明する．

■**[フルカスタム]**　フルカスタム IC は，決まった機能を最適な回路と配線によって実現した IC である．高い性能と安い製造単価を実現できる反面，長い開発期間と莫大な初期投資が必要である．このため，大量生産されるような用途 (例えば汎用 IC) に用いられる．

■[Gate Array]　Gate Array は，あらかじめ用意された雛形 (ゲート回路や特殊機能ブロックのライブラリと配線のアレイ) を用いて複雑な機能を短期間に開発できる IC である．多少の冗長性を持つため性能と製造単価はフルカスタムに劣る．しかし設計期間が短くでき初期投資もフルカスタムより安く済むため，個別のユーザーが自社の特定用途向け IC の開発に利用している．このことから ASIC (Application Specific IC) とも呼ばれる．

■[PLD / CPLD]　PLD (Programmable Logic Device) は，IC の内部配線をユーザー自身が変更できる (ユーザーが内部配線をプログラム可能な) 構造を持った論理 IC である．CPLD (Complex PLD) は内部に複数の PLD を持ち，より複雑な機能を実現できるようにした IC である．PLD や CPLD は単価が高く，性能もフルカスタムや Gate Array に比べて劣るが，初期投資の必要がなく書き換えも可能なため，論理回路の実験に用いたり，Gate Array 設計の初期段階で回路の論理をデバッグする目的で使用する．デバッグが終了した論理を製造単価の低い Gate Array へ移植することも多い．

■[FPGA]　FPGA は PLD と同様にユーザー自身がプログラムできる LSI であり，PLD よりも大規模かつ高速な回路を扱うことができる．名前の由来は「Field Programmable (現場でプログラム可能な) Gate Array」である．FPGA は単価が高いが，性能がよく初期投資も必要ない．PLD のように実験やデバッグに用いたり，非常に少量生産の製品で Gate Array では初期投資を回収できないような用途に用いられる．

通常，PLD は電源を切ってもプログラムされた配線情報は消えないが，一般的な FPGA は電源を切ると配線情報が消えてしまう．このため電源投入後に配線情報をロードしなければならない．この目的で ROM あるいは Flash ROM が利用されることも多いが，動的に回路を書き換えられる性質を積極的に利用した多目的な回路ボードも制作されている．このような FPGA の利用例として，本研究の中で開発された「回路とソフトウェアの協調計算システム」を 7 章で紹介している．

1.3.7　デジタル回路の設計方法

デジタル回路の設計手法は大きく二つに分類される．一つは**図面による設計**であり，もう一つは**言語による設計**である．言語による設計は現在の主流となっている．また，最近の研究として一階述語論理を用いた設計手法や書き換え規則を用いた設計手法がある．

図面による設計

- **トランジスタレベル**　トランジスタのレベルで回路を作成する．このレベルで設計された論理回路は，異なる製造工場 (fabrication) や異なるプラットフォーム (カスタム IC, ASIC, etc.) への移植が難しい．
- **ゲートレベル**　論理ゲートおよびフリップフロップレベルで回路図を作成する．トランジスタレベルより抽象度が上である．通常，基本ゲート (AND, OR, NOT) だけでなく「複合ゲート」と呼ばれるゲートも多用される．複合ゲートとは，複数の基本ゲートの組み合わせと同じ論理を，それら基本ゲートの組み合わせより少ないトランジスタ数で実現したゲート回路である．このため，複合ゲートを利用した組み合わせ回路は遅延時間と回路面積を小さくできる．ただし，利用可能な複合ゲートはメーカー，工場，プラットフォーム等によって異なるため可搬性はない．

```
module counter4(
    cnt ,
    reset ,
    clk
);

    output [3:0] cnt ;
    input        reset ;
    input        clk ;
    reg [3:0]    counter ;

    always @(posedge clk or posedge reset)
    begin
        if ( reset == 1'b1 ) begin
            counter <= 4'b0000 ;
        end else begin
            counter <= counter + 1 ;
        end
    end

    assign cnt = counter ;

endmodule
```

図 1.10　言語による回路設計の例：Verilog HDL による 4bit カウンター回路

言語による設計

言語による設計は図面による設計よりも抽象度が高く生産性が高いため，現在ではデジタル回路設計手法の主流となっている．

- **VHDL / Verilog HDL**　これらは現在主流のデジタル回路の設計用に開発された回路記述言語 (Hardware Description Language : HDL) である．VHDL[16] はアメリカ国防総省, Verilog HDL[17] はケイデンス社によって開発されたが，現在はいずれも IEEE で標準化されている．図 1.10 に Verilog HDL によるカウンター回路の設計例を示す．
  HDL で記述可能なものは大きく 3 つのレベルに分類できる．

1. **ビヘイビア (振る舞い) レベル**
2. **レジスタ転送レベル** (Register Transfer Level : **RTL**)
3. **ゲートレベル**

ビヘイビアレベルとは，プログラミング言語と同様に処理手順を記述する方法である．RTL とは，処理をクロック単位に分解して各クロック毎に発生する動作を記述する方法で，ビヘイビアより低位な設計方法である (図 1.10 は RTL レベル記述である)．ゲートレベルとは，ゲートおよびフリップフロップの接続関係 (ネットリスト) を記述する方法で，RTL よりもさらに低位な設計方法である．現在のところ**回路を生成可能な記述は RTL とゲートレベル**である．HDL で記述されたソースコードには可搬性があり，同じソースを異なる IC メーカーや異なる FPGA デバイスに利用できる．

- **SystemC / SystemVerilog**　SystemC[28] および SystemVerilog[29] は共にソフトウェアとハードウェアが混在するシステムを記述するための言語で，SystemC は C++ 言語を拡張したもの，SystemVerilog は Verilog HDL を拡張したものである．これらの言語で記述されたソースには可搬性がある．

最近の設計手法に関する研究

デジタル回路の設計規模の拡大によって，前述の言語による設計よりもさらに生産性の高い手法が望まれており，いくつかの研究がなされている．ここでは，その中から本研究に関連する，以下の二つを紹介する．

- **一階述語論理による設計の研究**　(P. Gaboury at University of Waterloo[30])
  本研究と同じく確定節によって宣言的仕様を記述する．この研究では確定節記述を直接回路へ変換している．このため回路の性能は宣言的仕様の「書き方」に大きく依存してしまう．本研究では ET ルールを経由することで様々な変換手法を利用できるようになっており，宣言的仕様の書き方に関わらず，効率のよい回路が生成できる．
- **書き換え規則による設計の研究**　(Hoe *et al.*[31, 32] , Rosenband[33, 34] at MIT)
  ガード条件付きの書き換え規則 [35] によって回路モジュールのビヘイビアを記述する．書き換え規則は個々の規則が独立であるため，各規則の正しさは他の規則に依存しない．したがって設計作業は，書き換え規則でアトミックな動作を個々に記述し，それらを集積すればよく，HDL に比べて容易である．しかし，MIT の研究は本研究のような「仕様」に対する正当性を保証する枠組みは持っていない．

1.3.8　SW と HW における記述言語の違い

SW 言語から実行バイナリを作る作業は “コンパイル” と呼ばれる．HW 言語から実行回路を作る作業は “論理合成 (synthesis)” と呼ばれる．

SW のソースコード

ソースコードは実行手順を表している．SW の言語はコンパイルすることで実行バイナリができるが，そのバイナリはプラットフォームによって大きく異なる．コンパイルの目的は命令列を得ることであり，利用可能な資源や命令セットの情報はコンパイラが持っている．SW プログラムのソースコードにはプラットフォーム (OS や CPU) の情報を記述する必要はないので可搬性がある．

HW のソースコード

ソースコードは回路の配線情報をあらわしている．HDL を論理合成して得られるものは回路の構造そのものである．

HW におけるプラットフォームの違いは，ASIC/FPGA/CPLD などの IC の実現方法や，埋め込み乗算器などのマクロの有無がある．これらプラットフォームに特有な埋め込み機能を使わなければ可搬性がある．

1.4 協調計算システムの概要

デジタル回路とソフトウェアにはそれぞれ長所と短所がある．協調計算システムは互いが欠点を補い，長所を活かして効率的な計算を実現するシステムである．

1.4.1 回路の長所と短所

一般にデジタル回路は単純な処理を並列に実行することに向いている．しかし，複雑な条件判定などが多用される処理にはデジタル回路の特徴を活かせない．また §1.3.3 で述べたように**メモリのアドレスという概念がなくポインタ参照が使えない**のも大きなデメリットである．

1.4.2 ソフトウェアの長所と短所

コンピュータはプログラミングにより複雑な計算でも柔軟かつ容易に実現することができる．また，デジタル回路に比べて計算可能なクラスが広い．しかしノイマンアーキテクチャを基本とするためメモリアクセスが局所的にしか行えず，処理はシーケンシャルに行われるため計算効率は悪い．

1.4.3 協調計算システムを実現する上での課題

■設計の問題　回路設計にはスキルが必要である．回路規模が増大している今日では回路の設計が非常に複雑化している．全体が並列に動作してしまうため，プログラムのような手続きの流れとして動作を捉えることができない．

協調計算システムの設計は更に複雑になる．回路設計のスキルとソフトウェア設計のスキルが異なることに加え，その両者を協調させて動作させるための制御が非常に難しい．

■正当性と検証の問題　現在の検証方法は，大量のテストケースをシミュレーションすることによりバグを発見するという手法である．シミュレーションによる検証では正当性を保証することが難しい．なぜなら，考えられる全てのシチュエーションをシミュレーションすることが不可能だからである．更に，回路とソフトの規模が増大するとシミュレーションすること自体に大きなコストがかかり，多くのテストケースを検証することは困難になる．

回路とソフトが異なる言語で設計されることも検証を難しくしている．例えば回路を Verilog HDL で記述し，ソフトを C++ 言語で記述する場合，これらの異なる言語を統合してシミュレーションすることは難しい．

1.5　本研究が目指す問題解決の枠組み

本研究は宣言的仕様レベルの設計 (高位設計) を目指す．この枠組みでは回路やシステムが宣言的仕様から理論的に生成され，正当性が数学的に保証される．

まず数学的な言語で記述された「宣言的仕様」と呼ばれる形式仕様を用いる．宣言的仕様から ET ルールを生成し，その ET ルールを変換して回路およびソフトウェアを生成する．宣言的仕様から生成される ET ルールは既存の研究である「等価変換理論」[1–9] により正当性が保証され，ET ルールの変換は本研究の成果である「代数系変換」(§5.2.1) によって正当性が保証される．

第 2 章

宣言的仕様，宣言的意味と正当性

本研究は，形式仕様として等価変換理論 [1–9] が採用している**宣言的仕様** (declarative specifications) を用いる．宣言的仕様の理解には一階述語論理 [36–38] および論理プログラミング [39–41] の知識が必要になる．

この章では宣言的仕様について簡単に説明する．まず最初に仕様記述言語として用いられる**確定節** (definite clause) について説明し，次に宣言的仕様と**宣言的意味** (declarative meaning) について説明したのち，最後に**正当性** (correctness) について説明する．

2.1 仕様記述言語としての確定節

確定節は従来から仕様記述や宣言的言語として用いられている．確定節については一階述語論理と論理プログラミングの文献 [36–41] に詳しいが，ここで簡単に説明する．

確定節 (論理プログラミングでは**プログラム節**ともいう) とは一階述語論理のサブセットであり，次の形式をした論理式である．

$$\text{確定節}: \ \forall \left(\neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_k \ \vee \ H\right) \qquad (k \geq 0) \tag{2.1}$$

ここで “ ¬ ” は論理否定 (NOT) をあらわす記号であり，“ ∨ ” は論理和 (OR) をあらわす記号である．また B_i (i=1, 2, . . . , k) および H は**原子論理式**をあらわす．原子論理式は**アトム** (atom) とも呼ばれる．一階述語論理のアトムは命題関数である．先頭の全称記号 “ ∀ ” は，式に現れるすべての変数が全称束縛されていることをあらわす．すなわち式 (2.1) の表記は，この式に現れる全ての変数を $X_1, X_2, \ldots, X_m$ としたときに下記の式と等価である．

$$^{\forall}X_1 \, ^{\forall}X_2 \ldots ^{\forall}X_m \left(\neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_k \ \vee \ H\right)$$

否定 “ ¬ ” のついたアトムを**負リテラル**と呼び，否定のつかないアトムを**正リテラル**と呼ぶ．確定節は，ただ一つの正リテラル H と 0 個以上の負リテラル $\neg B_i$ の**選言** (disjunction) である[*1]．

[*1] 一般にリテラルを論理和で連結した論理式を**節** (*clause*) と呼ぶ．特に正リテラルが 高々一つだけ の節を**ホーン節** (*Horn clause*)[42] と呼び，正リテラルが一つのホーン節を確定節と呼ぶ．

2.1.1 確定節の省略記法およびヘッドとボディ

確定節の式 (2.1) を同値変形すると下記の式が得られる．

$$\forall \left(B_1 \wedge B_2 \wedge \cdots \wedge B_k \ \rightarrow \ H \right)$$

式中の “ $\wedge$ ” は論理積 (AND) をあらわす記号であり，“ $\rightarrow$ ” は論理的含意 (implication) をあらわす記号である．すなわち確定節とはアトム間の関係

「アトム $B_1, \ldots, B_k$ が全て真ならば，アトム H は真である」

をあらわした式であることがわかる．**一般に確定節の表記方法は，上記の式を省略した下記の形式 (2.2) が用いられる**．本書でも同形式を用いる．

$$H \ \leftarrow \ B_1,\ B_2, \ldots,\ B_k\,. \tag{2.2}$$

省略形では全称記号 “ $\forall$ ” を省き，論理積 “ $\wedge$ ” をコンマ (,) で表す．本書では確定節の論理包含の矢印が**左向き** (“ $\leftarrow$ ”) となるように表記を統一している[*2]．矢印 “ $\leftarrow$ ” の左辺をヘッド，右辺をボディ と呼び，H を**ヘッドアトム**，各 B_i を**ボディアトム**と呼ぶ．

2.1.2 アトムと変数

アトム (式 (2.1) の B_i や H) は命題関数であり，任意個の引数を持っている．

$$\text{アトム：}\ p(t_1, \ldots, t_n) \qquad (n \geq 0)$$

一階述語論理では命題関数名 “ p ” を**述語名** (predicate name) と呼ぶ．本書では DEC–10 PROLOG[43] の文法に倣い，述語名には英小文字から始まるシンボルを用いる．例えば下記のアトムの述語名は add である．

$$\mathsf{add}(X, Y, 7) \tag{2.3}$$

アトムの引数 t_j (j=1, 2, …, n) には**項** (term) が入る．項には**変数** (variable) を含む式が許される．本書での変数の表記方法は，DEC–10 PROLOG の文法に倣って英大文字から始まるシンボルを用いる．例えば式 (2.3) の項 X, Y はそれぞれ変数である．また次に挙げるシンボルも有効な変数名の例である．

$$X1,\ XA,\ Xabc$$

変数には名前を持たない**無名変数** (anonymous variable) もあり，アンダースコア (_) であらわす．無名変数は他のどの変数とも名前が異なるものとして扱われる．たとえ一つの論理式の中に複数の無名変数が存在しても，それらは互いに異なる変数である．

[*2] **ET ルール** (3 章) の表記が確定節の表記に似ており，混同しないようにするためである．

基礎項，基礎アトム，基礎節

基礎項 (ground term) とは変数を含まない項 (定数記号と関数記号だけからなる項) のことである．そして **基礎アトム** (ground atom) とは引数が基礎項だけのアトムのことである．同様に，基礎アトムだけからなる確定節を**基礎節** (ground clause) と呼ぶ．次は基礎アトムの例である．

$$\mathsf{add}(2,5,7)$$

基礎項, 基礎アトム, 基礎節のことをそれぞれ**「グラウンドな項」**,**「グラウンドなアトム」**,**「グラウンドな節」**と呼ぶことがある．

基礎アトムは，決まった**解釈**の下でその真理値が真または偽に確定する．いま仮に，アトム $\mathsf{add}(\alpha,\beta,\gamma)$ を「$(\alpha+\beta)$ の値と γ の値は等しい」という意味に解釈すれば，上記の基礎アトムは真である．アトムが基礎アトムでないとき (引数に変数を含む項があるとき), そのアトムの真理値を確定することはできない．例えば $\mathsf{add}(2,Y,7)$ というアトムは変数 Y に代入される値によって真にも偽にもなるため Y の値が定まらなければアトムの真理値は分からない．

一方，**確定節の真理値は基礎節かどうかに関係しない**．確定節の真理値は，その確定節を構成している**アトムの解釈の仕方で決まる** (詳細は §2.1.5 で説明する)．逆に宣言的仕様では，確定節の記述からアトムの解釈を決定する (詳細は §2.3 で説明する).

2.1.3 アトムと述語

アトム (命題関数) の関数定義 (引数から { 真, 偽 } への写像) のことを**「述語の定義」**という．命題関数は**命題文**を記号化したものであり，命題文とは**主語**と**述語**に応じて真理値が決まる文である．命題関数の引数が文の主語に相当し，命題関数の写像 (真理値の決め方) が文の述語に相当している．

アトムが “意図通り” に解釈されるためには，アトムの述語を曖昧さなく明確に定義する方法が必要である．アトムは単なる記号にすぎず，アトムの名前 (すなわち述語名) とアトムの意味の間に命名規則はない．例えば前述の $\mathsf{add}(\alpha,\beta,\gamma)$ に対して何の情報も与えられなければ，あらゆる解釈の可能性がある．その可能性の中には例えば「α と β の平均値は γ の値に等しい」という解釈も存在する．

宣言的仕様は述語の定義を確定節の記述によって与えている (詳細は §2.3 で説明する). これによってアトムの解釈を一意に決めることが可能である．なお，同じ述語名を持つアトムでも引数の数 (arity) が異なるアトムは異なる命題文をあらわす．このため「述語」を特定するには下記のように述語名と arity の両方を指定する．

述語名/arity

例えば $\mathsf{add}(\alpha,\beta,\gamma)$ アトムの述語は下記に示す記号で特定される．

$$\mathsf{add}/3$$

2.1.4 変数と代入

論理式の変数に対する代入は次のような集合 θ によってあらわすのが一般的である．

$$\theta = \{ V_1/t_1,\ V_2/t_2,\ \ldots,\ V_m/t_m \} \tag{2.4}$$

代入 θ の各要素 “ V_i/t_i ” は，変数 V_i を項 t_i に置き換えることをあらわす[*3]．変数に対して別の変数を代入することも許される．例えば，次の例のような代入 θ_v は可能である．

$$\theta_v \ = \ \{X/Y,\ Y/Z\}$$

代入 θ_v は変数 X に変数 Y を，変数 Y に変数 Z を代入する．例えば θ_v をアトム $\mathsf{add}(X,Y,Z)$ に適用した結果を下記に示す．なお，代入の適用を後置記法であらわしている．

$$\mathsf{add}(X,Y,Z)\theta_v \ = \ \mathsf{add}(Y,Z,Z)$$

代入は再帰的には適用されない．すなわち，変数 X は “ X/Y ” が適用されて Y に置き換わるが，その結果にさらに “ Y/Z ” が適用されることはない．

$$\mathsf{add}(X,Y,Z)\theta_v \ \neq \ \mathsf{add}(Z,Z,Z)$$

基礎代入

変数 V_i を全て基礎項へ置き換える代入を**基礎代入** (ground substitution) と呼ぶ．例えば，$\mathsf{add}(X,Y,Z)$ に対して X=2，Y=5，Z=7 という代入は基礎代入であり，次の集合 θ_1 によってあらわされる．

$$\theta_1 = \{X/2,\ Y/5,\ Z/7\}$$

基礎代入 θ_1 をアトム $\mathsf{add}(X,Y,Z)$ に適用すると基礎アトムが得られる．

$$\mathsf{add}(X,Y,Z)\theta_1 \ = \ \mathsf{add}(2,5,7)$$

論理式の変数と手続き型プログラミング言語の変数の違い

手続き型のプログラミング言語 (C/C++, Java, etc.) における「変数」とは値を格納する場所 (ストレージ) であり，何度でも内容を書き換えることができる．一方，論理式における変数はストレージではなく，値につけられた「ラベル」ような働きをする．論理式の変数への代入は，変数の値を確定させることである (ただし別の変数が代入される場合は値が確定しない)．値が確定した変数に対して「上書き」や「再代入」は発生しない．

[*3] この表記は参考文献 [39] の流儀である．“ t_i/V_i ” と表記する流儀も存在する．

2.1.5 確定節の真理値

確定節の先頭の全称記号 “ ∀ ” は式に現れるすべての変数が全称束縛されていることをあらわす．したがって，確定節が真であるとは，いかなる基礎代入 θ に対しても次の論理式が真となることである．

$$(\neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_k \ \vee \ H)\theta$$

すなわち，任意の基礎代入 θ に対してリテラル ($\neg B_i\theta$ または $H\theta$) の中に真となるものが存在していればよい．例として，アトム $\mathsf{add}(\alpha,\beta,\gamma)$ が「$(\alpha+\beta)$ の値と γ の値は等しい」という意味の命題関数のとき，次の確定節が真であることを説明する．

$$\forall\Big(\ \neg\mathsf{add}(X,Y,6) \ \vee \ \neg\mathsf{add}(X,X,Y) \ \vee \ \mathsf{add}(X,5,7)\ \Big) \tag{2.5}$$

この確定節 (2.5) の負リテラル $\neg\mathsf{add}(X,Y,6)$ および $\neg\mathsf{add}(X,X,Y)$ が共に偽になる基礎代入は次の θ_a だけである (連立方程式 { X+Y=6，X+X=Y } の解).

$$\theta_a \ = \ \{X/2,\ Y/4\}$$

つまり θ_a 以外の任意の基礎代入 θ' $(\neq \theta_a)$ に対して確定節 (2.5) の負リテラルのいずれかが真となる．

$$\forall\theta' \ (\neq \theta_a) \text{ に対して } \ \neg\mathsf{add}(X,Y,6)\theta' \ \text{ または } \ \neg\mathsf{add}(X,X,Y)\theta' \ \text{ が真} \tag{2.6}$$

一方，基礎代入 θ_a に対しては明らかに正リテラルが真となる．

$$\theta_a \text{ に対して } \ \mathsf{add}(X,5,7)\theta_a \ \text{ が真} \tag{2.7}$$

条件 (2.6) および (2.7) から，任意の基礎代入 θ (θ_a を含む) に対していずれかのリテラルが真となるので下記の論理式は真である．

$$(\ \neg\mathsf{add}(X,Y,6) \vee \neg\mathsf{add}(X,X,Y) \vee \mathsf{add}(X,5,7)\ )\,\theta \tag{2.8}$$

いかなる基礎代入 θ に対しても上式が真なので確定節 (2.5) は真なる確定節である．

ところで，**アトムを異なった意味に解釈すると確定節が真にならない**場合がある．確定節 (2.5) のアトムの意味が間違って解釈された場合，例えば $\mathsf{add}(\alpha,\beta,\gamma)$ に対して「α の値と $(\beta+\gamma)$ の値は等しい」という解釈がなされたとき，確定節 (2.5) は偽である[*4]．アトムの意味が**意図どおり**に解釈されなければ，論理に矛盾を生じる可能性がある．このため**アトムの解釈 (述語定義) を明確にしなければならない**．宣言的仕様の中ではアトムの述語定義は確定節によって記述され，その解釈は機械的に一意に決定される．これは**宣言的意味** (§2.3) と呼ばれる．以降では宣言的仕様，宣言的意味，正当性について説明する．

[*4] 反例 X=6, Y=0 が存在する．

2.2 ET における宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$

ET の形式仕様は確定節によって記述される**宣言的仕様**である．宣言的仕様に必要なことは，どのような問題を解くのかという「質問の定義」と，問題を解くために必要な「背景知識の定義」である．具体的には問題の背景知識の集合 $\mathbb{D}$ と質問の集合 $\mathbb{Q}$ をそれぞれ確定節によって記述する．ET では，この組 $\langle \mathbb{D}, \mathbb{Q} \rangle$ のことを宣言的仕様と定義している．実現されるべきシステムの動作は宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$ によって決定される．このことを階乗の計算を例に用いて説明する．

2.2.1 背景知識 $\mathbb{D}$

問題を解くためには，その問題に関する背景知識が必要となる．例えば，階乗の値を求めるには「階乗とは何か」についての知識が必要であり，その知識は一般に下記の再帰的な式によって与えられる．

$$\begin{aligned} 0! &= 1 \\ n! &= n \times (n-1)! \end{aligned}$$

宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$ では背景知識 $\mathbb{D}$ を「確定節」を用いて記述する．確定節は「アトム間の関係」を記述する論理式なので，まず「何らかの関係」をアトムで表現する必要がある．ここではアトム factorial(m, n) を用いて「階乗の関係」を表現することにする．しかし，このアトム factorial(m, n) がどのように階乗の関係を表現しているのか，その解釈の仕方は自明ではない．ここで，このアトムが誤解なく一意に解釈されるには，数学的な言葉で明確に記述された「説明」が必要である．それが下記の背景知識 $\mathbb{D}_f$ である．

$$\mathbb{D}_f = \left\{ \begin{array}{l} \mathsf{factorial}(0, 1) \leftarrow . \\ \mathsf{factorial}(N, N \times F) \leftarrow \mathsf{factorial}(N-1, F). \end{array} \right\}$$

ここで N および F は変数を表している．この背景知識 $\mathbb{D}_f$ は前述の再帰的な階乗の定義式を確定節によって記述したものであり，アトム factorial(m, n) に対して「m の階乗は n である」という解釈を与えている．

2.2.2 質問の集合 $\mathbb{Q}$

質問の集合 $\mathbb{Q}$ とは「求めるものは何か，どの範囲まで計算するのか」を定義する．まず具体的な質問例を個々に定義することで求めるものが決まり，これら個々の質問の集合によって計算範囲が決まる．

個別の質問をあらわす ans 節

個々の具体的な質問例を定義するものが ans 節 である．ans 節は **ans アトム**と呼ばれる特別なアトムをヘッドに持つ確定節である[*5]．例えば，下記の ans 節は「5 の階乗の値を求める」という具体的なひとつの質問をあらわす．

$$\mathsf{ans}(X) \leftarrow \mathsf{factorial}(5, X).$$

ここで X は変数をあらわしている．この確定節は「X の値が 5 の階乗ならば，それが求める答え (ans) である」という意味の論理式である．上記 ans 節と背景知識 $\mathbb{D}_f$ から ans(120) が真であるという結論が得られる．この結論は質問の答え (5! = 120) になっている．

ans 節の集合で計算範囲が定まる

質問の集合 $\mathbb{Q}$ は ans 節の集合として記述され，何を求めるのか，どの範囲まで計算するのかを定義する．例えば，求めるものが階乗の値であり，計算範囲が自然数全体 $\mathbb{N}$ のとき，質問の集合は次の $\mathbb{Q}_1$ のように定義される．

$$\mathbb{Q}_1 = \left\{ \mathsf{ans}(X) \leftarrow \mathsf{factorial}(n, X) \mid n \in \mathbb{N},\ X \in \mathbb{V} \right\}$$

ここで $\mathbb{V}$ は変数の集合である．階乗の背景知識 $\mathbb{D}_f$ と上記の集合 $\mathbb{Q}_1$ による宣言的仕様 $\langle \mathbb{D}_f, \mathbb{Q}_1 \rangle$ が与えられたとき，この宣言的仕様に対して正当なシステムは任意の自然数に対する階乗の計算を実行する．

同じ背景知識に対して異なる質問が可能

アトム factorial(α, β) は α と β の関係を述べているにすぎないので階乗の逆関数を計算する ans 節も定義できる．例えば階乗して 720 になる数を求める ans 節は次のように定義される．

$$\mathsf{ans}(X) \leftarrow \mathsf{factorial}(X, 720).$$

この ans 節と背景知識 $\mathbb{D}_f$ から答え ans(6) が結論される．そして計算する範囲が自然数全体の場合，質問の集合は次の $\mathbb{Q}_2$ のように定義される．

$$\mathbb{Q}_2 = \left\{ \mathsf{ans}(X) \leftarrow \mathsf{factorial}(X, n) \mid n \in \mathbb{N},\ X \in \mathbb{V} \right\}$$

この場合には質問に対して答えが存在しないことがある．その場合「計算が失敗 (fail) する」という．

[*5] ans 節のヘッドアトム (ans) は，背景知識 $\mathbb{D}$ の中に出現してはいけない．

2.2.3 宣言的仕様と論理的帰結

宣言的仕様が定義する問題を解くということは，質問の ans 節と背景知識からの**論理的帰結**を求めることである．例えば上記の「階乗して 720 となる数を求める」という ans 節の答え ans(6) は次の関係を満たすように求められたものである．

$$\Big\{ \mathsf{ans}(X) \leftarrow \mathsf{factorial}(X, 720) \Big\} \cup \mathbb{D}_f \quad \models \quad \mathsf{ans}(6)$$

ここで記号 "$\models$" は論理的帰結をあらわす．論理的帰結とは，左辺の論理式 (ans 節と $\mathbb{D}_f$) が真のとき右辺の論理式 (ans(6)) が必ず真である (偽となる反例が存在しない) という論理的な充足関係のことである．

宣言的仕様には解き方を記述する必要はない．宣言的仕様はアルゴリズムや実装方法をシステムに任せることを前提としており，答えが満たすべき充足関係が宣言的に記述されていればよい．宣言的な記述によって問題のロジックが分かりやすくなり，仕様を記述する上で非常に大きな利益をもたらす．

2.2.4 宣言的仕様を満たすアルゴリズム

宣言的仕様は解き方を与えてくれないため，どのように答えを導出するのかが大きな課題となる．例えば，あらゆる値に対して論理的帰結かどうかを判定しながら答えを探索するのも一つの方法である．多くの場合には計算量が莫大になり実用的ではないかもしれないが，探索範囲が狭い場合には最も簡単に実現できる．

一階述語論理には，**導出原理**と呼ばれる推論規則に基づく万能なアルゴリズムが存在し，必ず答えを導出できることが示されている [39–41]．そのようなアルゴリズムのひとつに PROLOG などの処理系に採用されている ***SLD*–反駁** [44, 45] がある[*6]．しかし SLD–反駁は答えを探索する過程でバックトラックを生じるので効率的な計算ができないという欠点を持っている．またデジタル回路でバックトラックを実装することは難しい．

等価変換理論では，宣言的仕様に応じた効率的なアルゴリズムを ***ET* ルール** と呼ばれる書き換え規則の集合として自動生成する研究が幾つか与えられており [7–12]，現在も研究が進められている．ET ルールで記述されたアルゴリズムはバックトラックを生じない．また，書き換え規則を適用する条件を細かく制御できるため無駄な探索を減らすことができる．

2.3 宣言的意味

正当性を論じるために確定節集合に対して形式的に「意味」を定義する．「意味」は形式的なものであり，その定義方法はさまざま考えられる．ここでは**宣言的意味**という定義を

[*6] PROLOG は計算効率向上のため SLD–反駁アルゴリズムを簡略化しているので完全性を満たさない．

用いる．

この章では，まず従来の論理プログラミングで用いられている宣言的意味の定義を説明する．次に，それを拡張した等価変換理論における「意味」の定義を説明する．等価変換理論では，項 (データ構造) を拡張した “制約付き” の確定節を扱えるように**特殊化システム** [46–48] と呼ばれる仕組みを導入しており，宣言的意味の定義も拡張されている．ただし，本書では制約付き確定節についての説明は省略する (参考文献 [47]).

2.3.1 最小エルブランモデルによる意味 $M_{\mathbb{P}}$ の定義

確定節集合 $\mathbb{P}$ が与えられているとき，その $\mathbb{P}$ の形式的意味を $M_{\mathbb{P}}$ と表記することにする．従来の論理プログラミングでは確定節集合の形式的な意味 $M_{\mathbb{P}}$ を **最小エルブランモデル** として与えている [39, 40].

最小エルブランモデル (least Herbrand model)

確定節集合 $\mathbb{P}$ の最小エルブランモデルとは，$\mathbb{P}$ からの論理的帰結であるすべての基礎アトムからなる集合である．最小エルブランモデルが形式的な「意味」として妥当であることを次の確定節集合 $\mathbb{P}_1$ を用いて直観的に説明する．$\mathbb{P}_1$ は自然数の奇数と偶数の知識を記述したものである．

$$\mathbb{P}_1 = \left\{ \begin{array}{l} \mathsf{even}(0) \leftarrow . \\ \mathsf{odd}(X+1) \leftarrow \mathsf{even}(X). \\ \mathsf{even}(X+1) \leftarrow \mathsf{odd}(X). \end{array} \right\}$$

この確定節集合に対する最小エルブランモデル (すなわち $\mathbb{P}_1$ の論理的帰結であるすべての基礎アトムからなる集合) は次の集合 $\mathbb{A}_1$ で与えられる．

$$\mathbb{A}_1 = \left\{ \begin{array}{c} \mathsf{even}(0),\ \mathsf{odd}(1) \\ \mathsf{even}(2),\ \mathsf{odd}(3) \\ \mathsf{even}(4),\ \mathsf{odd}(5) \\ \vdots \end{array} \right\}$$

基礎アトム集合はアトムの**解釈**[*7] を定める役目をする．解釈とは基礎アトムの真理値を決めることである．この集合は，真なる基礎アトムのすべてを外延的に列挙したものであり，この集合に含まれない基礎アトムは偽とみなされる．$\mathbb{A}_1$ の解釈によれば，$\mathsf{even}(n)$ は「n が偶数」という意味を，$\mathsf{odd}(n)$ は「n が奇数」という意味をあらわしており，$\mathbb{P}_1$ の「意図したとおりの」解釈になっている．

一般に確定節集合 $\mathbb{P}$ の最小エルブランモデルは $\mathbb{P}$ の意図した解釈を与える．したがって $M_{\mathbb{P}}$ = 「$\mathbb{P}$ の最小エルブランモデル」と定義するのは妥当である．

[*7] 厳密には，ここでの解釈は**エルブラン解釈** [39–41] と呼ばれるものである．

最小エルブランモデルでないモデル

先の例で示した $\mathbb{A}_1$ は $\mathbb{P}_1$ を**充足**する (つまり $\mathbb{A}_1$ の解釈の下では $\mathbb{P}_1$ の全ての論理式が真である)．このような解釈を「モデル」と呼ぶ．最小エルブランモデルは確定節集合に対する最小のモデルであることが示されている [39, 40]

最小エルブランモデルでないモデルは意図した解釈を与えない．例えば，次の基礎アトム集合 $\mathbb{A}_2$ は $\mathbb{P}_1$ を充足するので「モデル」であるが意図した解釈を与えない．

$$\mathbb{A}_2 = \left\{ \begin{array}{c} \mathsf{even}(0),\ \mathsf{odd}(0) \\ \mathsf{even}(1),\ \mathsf{odd}(1) \\ \mathsf{even}(2),\ \mathsf{odd}(2) \\ \vdots \end{array} \right\}$$

解釈 $\mathbb{A}_2$ には $\mathbb{P}_1$ の論理的帰結ではない基礎アトム (例えば，$\mathsf{odd}(0)$，$\mathsf{even}(1)$，$\mathsf{odd}(2)$ など) が含まれている．このため意図した解釈になっていないのである．

2.3.2 意味 $M_{\mathbb{P}}$ の帰納的計算式

任意の確定節集合 $\mathbb{P}$ の最小エルブランモデル (すなわち意味 $\boldsymbol{M}_{\mathbb{P}}$) は帰納的に計算できることが示されている (参考文献 [39–41])．ここでは，$\boldsymbol{M}_{\mathbb{P}}$ が帰納的に計算できることを簡単に説明し，その計算式を紹介する．これは，次節 (§2.3.3) で「等価変換理論における意味の定義」を説明するための布石でもある．

$\boldsymbol{M}_{\mathbb{P}}$ を帰納的に計算する式

確定節集合 $\mathbb{P}$ の最小エルブランモデルは，$\mathbb{P}$ の論理的帰結であるすべての基礎アトムの集合である．つまり $\boldsymbol{M}_{\mathbb{P}}$ は次の数式であらわされる．

$$\boldsymbol{M}_{\mathbb{P}} \ = \ \{\, \alpha \in \boldsymbol{B}(\mathbb{P}) \mid \mathbb{P} \models \alpha \,\} \tag{2.9}$$

ここで $\boldsymbol{B}(\mathbb{P})$ は **エルブラン基底** (Herbrand base) と呼ばれる基礎アトム集合をあらわしている．エルブラン基底 $\boldsymbol{B}(\mathbb{P})$ とは，確定節集合 $\mathbb{P}$ に出現するアトムにあらゆる基礎項を組み合わせて作ることのできる，あらゆる基礎アトムの集合である．

上式 (2.9) の右辺の集合は，ある写像 $T_{\mathbb{P}} : \mathbf{2}^{\boldsymbol{B}(\mathbb{P})} \to \mathbf{2}^{\boldsymbol{B}(\mathbb{P})}$ の最小不動点に一致することが知られている．また，$T_{\mathbb{P}}$ は **完備束** $\langle 2^{\boldsymbol{B}(\mathbb{P})}, \subseteq \rangle$ 上で定義された写像であり，なおかつ単調で連続という性質を持つ．このような性質を持つ写像の最小不動点は帰納的に求められることが文献 [39–41] に示されている．下記にその帰納的計算式を紹介する．

$\boldsymbol{M}_{\mathbb{P}}$ の帰納的計算式

まず記号を定義する．$\mathbb{P}$ を確定節集合，$\boldsymbol{S}$ を全ての基礎代入からなる集合，$\boldsymbol{B}(\mathbb{P})$ を $\mathbb{P}$ のエルブラン基底とする．任意の確定節 $s \in \mathbb{P}$ に対し $\mathsf{head}\,(s)$ は節 s のヘッドアトムを返す関数，$\mathsf{body}\,(s)$ は節 s のボディ(ボディアトムの集合) を返す関数とする．次に，これらの定義された記号を用いて任意の確定節集合 $\mathbb{P}$ と任意の基礎アトム集合 $\boldsymbol{I} \subseteq \boldsymbol{B}(\mathbb{P})$ に対する $\mathbf{2}^{\boldsymbol{B}(\mathbb{P})}$ 上の写像 $T_{\mathbb{P}} : \mathbf{2}^{\boldsymbol{B}(\mathbb{P})} \to \mathbf{2}^{\boldsymbol{B}(\mathbb{P})}$ を次のように定義する．

$$T_{\mathbb{P}}(\boldsymbol{I}) \stackrel{\text{def}}{=} \{\, \mathsf{head}\,(s\theta) \in \boldsymbol{B}(\mathbb{P}) \mid (s \in \mathbb{P}) \wedge (\theta \in \boldsymbol{S}) \wedge (\mathsf{body}\,(s\theta) \subseteq \boldsymbol{I}) \,\}$$

このとき確定節集合 $\mathbb{P}$ の論理的帰結となる基礎アトム全体からなる集合 $\boldsymbol{M}_{\mathbb{P}}$ は写像 $T_{\mathbb{P}}$ の最小不動点に一致し，それは次式で計算できる．

$$\boldsymbol{M}_{\mathbb{P}} = \bigcup_{n=1}^{\infty} [T_{\mathbb{P}}]^n(\emptyset) \tag{2.10}$$

where $[T_{\mathbb{P}}]^1(\emptyset) = T_{\mathbb{P}}(\emptyset)$,
$[T_{\mathbb{P}}]^{n+1}(\emptyset) = T_{\mathbb{P}}\left([T_{\mathbb{P}}]^n(\emptyset)\right) \quad (n = 1, 2, \ldots)$

2.3.3 等価変換理論における意味 $\mathcal{M}(\mathbb{P})$ の定義

等価変換理論における意味は，従来の意味の定義から拡張されている．従来の意味 $\boldsymbol{M}_{\mathbb{P}}$ と区別するため等価変換理論における意味を $\mathcal{M}(\mathbb{P})$ と表記する．

等価変換理論では，従来の論理プログラミングの「最小エルブランモデル」のような直観的に理解しやすい「意味」の定義は与えられていない．これは，項 (データ構造) を拡張した “制約付き” の確定節を扱えるように**特殊化システム** [46–48] と呼ばれる仕組みを導入しているためである．その代り $\boldsymbol{M}_{\mathbb{P}}$ の帰納的計算式 (2.10) を自然に拡張した式を用いて $\mathcal{M}(\mathbb{P})$ を定義している．次に，その定義式を紹介する．

Definition 2.3.1. ET における意味の定義
与えられた特殊化システムを Γ とする．以下は全て Γ 上で定義されたものであるとする．
まず記号を定義する．$\mathbb{P}$ を確定節集合，$\mathcal{S}$ を全ての基礎代入からなる集合，$\mathcal{G}$ を全ての基礎アトムからなる集合とする．任意の確定節 $s \in \mathbb{P}$ に対し $\mathsf{head}(s)$ は節 s のヘッドアトムを返す関数，$\mathsf{body}(s)$ は節 s のボディアトム集合を返す関数とする．
次に，これらの定義された記号を用いて確定節集合 $\mathbb{P}$ と任意の基礎アトム集合 $G \subseteq \mathcal{G}$ に対する $\mathbf{2}^{\mathcal{G}}$ 上の写像 $T_{\mathbb{P}} : \mathbf{2}^{\mathcal{G}} \to \mathbf{2}^{\mathcal{G}}$ を次のように定義する．

$$T_{\mathbb{P}}(G) \overset{\text{def}}{=} \{\, \mathsf{head}(s\theta) \in \mathcal{G} \mid (s \in \mathbb{P}) \wedge (\theta \in \mathcal{S}) \wedge (\mathsf{body}(s\theta) \subseteq G) \,\}$$

このとき確定節集合 $\mathbb{P}$ の意味 $\mathcal{M}(\mathbb{P})$ は次式で 定義される．

$$\mathcal{M}(\mathbb{P}) \overset{\text{def}}{=} \bigcup_{n=1}^{\infty} [T_{\mathbb{P}}]^n(\emptyset) \tag{2.11}$$

$$\text{where } [T_{\mathbb{P}}]^1(\emptyset) = T_{\mathbb{P}}(\emptyset),$$
$$[T_{\mathbb{P}}]^{n+1}(\emptyset) = T_{\mathbb{P}}\left([T_{\mathbb{P}}]^n(\emptyset)\right) \quad (n = 1, 2, \ldots)$$

本書では制約付きの確定節について説明しない (参考文献 [47])．扱う例題も通常の論理プログラミングの範囲であるが，等価変換理論は論理プログラミングを包含しており，論理プログラミングの正当性を等価変換理論上で議論できる[*8]．本研究が回路記述に用いている **ET ルール**は，$\mathcal{M}(\mathbb{P})$ を用いた等価変換理論上で正当性が保証されている．

2.4 正当性

2.4.1 論理プログラミングの正当性

論理プログラミングにおける計算とは，ゴール節と呼ばれる論理式

$$\leftarrow B.$$

と確定節集合 $\mathbb{D}$ から**導出規則**を用いて空節 (矛盾) を導くことである．ゴール節は Horn 節 [42] の一種であり質問の否定 ($\neg B$) をあらわしている．導出規則とは，その正当性が証明されている汎用の演繹規則である．つまり論理プログラミングの計算とは，導出規則による反駁手続き (背理法による定理証明) であり，その例証としてアトム B への解代入 θ が得られる．

[*8] このことは，定義式 (2.11) において，基礎代入 $\mathcal{S}$ を $\boldsymbol{S}$ に，基礎アトム集合 $\mathcal{G}$ を $\boldsymbol{B}(\mathbb{P})$ に置き換えると $\boldsymbol{M}_{\mathbb{P}}$ と同じになることからわかる

論理プログラミングの正当性は導出規則の健全性に基づいている．すなわち，導出規則による質問 B の反駁手続きが成功すると，得られた解代入 θ が必ず次式を満たす．

$$B\theta \in M_{\mathbb{P}}$$

導出規則に基づく反駁手続きは数多く存在する [39]．特に *SLD*-**反駁** [44, 45] と呼ばれるアルゴリズムが有名であり，PROLOG などの処理系に採用され広く利用されている．

2.4.2 ET 計算モデルの正当性

ET 計算モデルにおける計算は，質問節 $s_0 \in \mathbb{Q}$ を書き換え規則によって次々に書き換えて**単位節** (ボディアトムが 0 個の確定節) を導くことである．

$$s_0 \mapsto s_1 \mapsto \cdots \mapsto s_{i-1} \mapsto s_i \mapsto \cdots \mapsto s_n.$$

最終的に得られる節 s_n が単位節ならば，そのヘッド (ans アトム) が求める答えである．正当性を満たす書き換え規則は，次の条件を満たす任意のものが許される．

Definition 2.4.1. ET ルールの条件

宣言的仕様を $\langle \mathbb{D}, \mathbb{Q} \rangle$ とする．節に対する書き換え規則 r を考える．この r は節 s_{i-1} を節 s_i に書き換えるものとする．このとき，r は次の式を満たさなければならない．

$$\mathcal{M}(\mathbb{D} \cup \{s_{i-1}\}) = \mathcal{M}(\mathbb{D} \cup \{s_i\}) \tag{2.12}$$

この関係を満たす書き換えを**等価変換** (Equivalent Transformation : ET) と呼び，等価変換を行う書き換え規則 r を *ET* **ルール**と呼ぶ．

「等価変換 (Equivalent Transformation)」とは「意味」を変えない書き換えのことである．背景知識の集合 $\mathbb{D}$ を明示しなくても誤解が生じない場合には $\mathcal{M}(\mathbb{D} \cup \{s\})$ の $\mathbb{D}$ を省略して簡単に $\mathcal{M}(s)$ と記述することにする．

$$\mathcal{M}(s_{i-1}) = \mathcal{M}(s_i) \tag{2.13}$$

書き換え規則として ET ルールだけを用いて計算するとき，任意の i $(1 \le i \le n)$ に対して常に式 (2.13) が成り立つので，質問の節 $s_0 \in \mathbb{Q}$ と答えの節 s_n の間に次の関係が成り立つことは容易に理解できる．

$$\mathcal{M}(s_0) = \mathcal{M}(s_n) \tag{2.14}$$

これは質問 s_0 と答え s_n の間で「意味」が保存されていることをあらわす．このように，ET ルールは常に式 (2.13) を満たしているので計算の正当性が容易に保証されるのである．

ET ルールの生成

論理プログラミングでは質問 (ゴール節) から空節を導出するため，$\mathbb{D}$ の確定節とゴール節から新たな論理式を導出する演繹を行っている．このため $\mathbb{D}$ をプログラムと考える．一方，等価変換計算モデルでは $\mathbb{D}$ は式 (2.12) を満たすような ET ルールの生成に使われるだけであり，実際の計算は ET ルールによる節の書き換えである．どうやって ET ルールを作成するのかは重要な課題であるが，本書の範囲ではないので説明を省略する．これらについては宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$ から自動生成する研究が幾つか与えられており [7–12]，また現在も研究が進められているところである．

2.4.3 宣言的意味論と手続き的意味論

通常のプログラミングではプログラマーは手続きを気にして記述する．しかし，最初に存在するのは宣言的仕様であり宣言的仕様を満たすプログラムを作成することがプログラミングにとって重要である．

論理プログラミングではプログラムを論理式の集合 P として考え，与えられた宣言的仕様 (解釈 I) に対して $\mathcal{M}(P)=I$ を満たすような論理式集合 P を作成することがプログラミングであると考える．

論理プログラミングにおける計算結果は確定節集合からの論理的帰結である．論理的帰結とは充足関係を述べているにすぎず，その計算方法には言及しない．もし，確定節の集合が充足関係を定義しているととらえれば，それは宣言的意味論になる．

宣言的意味論は実際の計算方法は与えてくれないため，導出原理 (MP, SLD, ETR など) が必要である．導出原理に基づく証明過程を計算ととらえれば，これは手続き的意味論になる．

そして宣言的意味論と手続き的意味論の間に完全性が成り立つなら，宣言的に記述されたプログラムを導出原理に基づいて手続き的に解くことができる．SLD は，そのような完全性が証明された手続きであり，宣言的仕様で与えられた問題の「解き方」が存在することを保証してくれる．

第 3 章

等価変換と ET ルール

本研究が用いるソースコードの記述言語は *ET* **ルール**と呼ばれる書き換え規則である．ET ルールは**等価変換理論** [1–9] によって宣言的仕様 (2 章) との間に数学的な正当性関係が保証されるという特徴を持つ．また，宣言的仕様から ET ルールを自動生成する研究も多数おこなわれている [7–12]．この章では等価変換および ET ルールについての説明を行う．

3.1 等価変換 (ET) の紹介

宣言的記述による**等価変換** (Equivalent Transformation : ET) 計算は，最初に実験的に自然言語理解システムに組み込まれた (文献 [1, 2])．その後，一階述語論理の項を扱う領域で広く使用され，更に RDF,UML,XML を含む幾つかのデータ領域において ET 計算モデルは知識推論システムに応用されている (文献 [49–51])．

等価変換の枠組みの中ではプログラムの作成に関して，正当な ET プログラム (ET ルールの集合) は，与えられた宣言的仕様から系統だった方法で作り出すことができる．そして他の計算モデルと比べ，正当なプログラムの計算効率を正当性を維持したまま改善することが低コストで可能である．また，等価変換理論に基づいて生成される ET ルールは**プログラム変換**が容易であるという特徴を持つ．

3.2 ET ルールと計算

論理プログラミングの計算では，導出規則に基づいて質問 (ゴール節) を確定節で書き換えている．これに対し，等価変換による計算では *ET* **ルール**と呼ばれる書き換え規則を用いた節の計算モデルを提案している．ET ルールは書き換え規則 [35] の一種で，質問の確定節 (ans 節) を書き換える．この書き換えの正当性は等価変換理論によって保証される．ET ルールによる計算ではバックトラックを生じない．またコンテクストに応じた制御が可能なので SLD に比べて効率の良い計算が可能などの特徴がある．

3.2.1 ET ルールの例

Example 3.2.1. ET ルールの例

$$\mathsf{factorial}(N, X), \{N > 0\} \;\texttt{==>}\; \{N1 := N - 1\},\ \mathsf{factorial}(N1, X1),\ \mathsf{mul}(N, X1, X). \tag{3.1}$$

$$\mathsf{factorial}(N, X), \{N = 0\} \;\texttt{==>}\; \{X := 1\}. \tag{3.2}$$

$$\mathsf{mul}(X, Y, Z), \{(X \in \mathbb{N}) \wedge (Y \in \mathbb{N}) \wedge (Z \in \mathbb{V})\} \;\texttt{==>}\; \{Z := X \times Y\}. \tag{3.3}$$

ET ルールは確定節を書き換える**書き換え規則**であり，矢印記号 " ==> " は「左辺のアトム集合を右辺のアトム集合に置き換える」ことをあらわしている．本書では ET ルールの矢印の表記を右向き記号 " ==> " に統一する．これは論理式の含意記号 (" → " や " ⇒ ") との混同を避けるためである．

矢印 " ==> " の左辺のアトム集合を**ヘッド**，右辺のアトム集合を**ボディと**呼ぶ．確定節でも含意記号 "←" の両辺をヘッド/ボディと呼ぶため，区別が必要な場合には**ルールヘッド/ルールボディ**と呼ぶことにする．なお，**確定節のヘッドは一個のアトム**であるが，**ルールヘッドはアトム集合**である．また，ET ルールと確定節では矢印の向きに対するヘッドとボディの位置が逆になっているので注意が必要である．

矢印 " ==> " の左辺の { } を**条件部**，右辺の { } を**実行部**と呼ぶ．条件部の中は論理式，実行部の中は手続き (計算式や代入式) が入る．ET ルールは条件部が真のときに書き換え可能であり，書き換えが適用されるときに実行部が実行され節の変数に代入が生じる．

3.2.2 ET ルールによる書き換えの仕組み

例として，次の確定節 (ans 節) を例 3.2.1 の ET ルールで書き換える仕組みを説明する．

$$\mathsf{ans}(F) \;\leftarrow\; \mathsf{factorial}(2, W),\ \mathsf{mul}(3, W, F).$$

まず ans 節に**適用可能**な ET ルールを見つける．適用可能とは，適当な代入 θ によって ET ルールのヘッドが ans 節のボディの**部分集合**になり，なおかつ条件部が真となることである．この例では ET ルール (3.1) だけが適用可能である．ET ルール (3.1) が適用可能であることは，次の代入 θ の存在によってわかる．

$$\theta \;=\; \{N/2,\ X/W,\ N1/T,\ X1/G\}$$

この代入 θ を ET ルール (3.1) に適用した結果，下記のルールが得られる．

$$\mathsf{factorial}(2, W),\ \{2 > 0\} \;\texttt{==>}\; \{T := 2 - 1\},\ \mathsf{factorial}(T, G),\ \mathsf{mul}(2, G, W)$$

代入 θ 適用後のルールのヘッドアトム集合 $\{\mathsf{factorial}(2, W)\}$ が ans 節のボディの部分集合になっており，且つ条件部 $(2 > 0)$ が真なので ET ルール (3.1) は ans 節に適用可能である．

なお，代入 θ の最後の二つの要素は**変数名変更代入**と呼ばれ，計算中に変数名が衝突しないように ET ルールの変数名をユニークに変更してから節へ適用する働きをしている．

ans 節のボディのうち，ET ルールのヘッドにマッチした $\{\,\mathsf{factorial}(2, W)\,\}$ の部分が ET ルールのボディに置き換えられ，次の節が得られる．

$$\mathsf{ans}(F) \;\leftarrow\; \underline{\mathsf{factorial}(T, G),\ \mathsf{mul}(2, G, W),}\ \mathsf{mul}(3, W, F).$$

次に ET ルールの実行部 $\{T := 2 - 1\}$ が実行され，変数 T への代入 ρ=$\{T\,/\,1\}$ が得られる．得られた代入 ρ は ans 節に適用され，最終的に次のように書き換えられる．

$$\mathsf{ans}(F) \;\leftarrow\; \mathsf{factorial}(1, G),\ \mathsf{mul}(2, G, W),\ \mathsf{mul}(3, W, F).$$

3.2.3 ET による計算の例

上記の例に示した ET ルールによって「3 の階乗」を計算する過程を下記に示す．なお，見やすさのため書き換え対象のアトムには下線を引いている．

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{factorial}(3, X)}.$

↓ *rewritten by rule* (3.1)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{factorial}(2, X11)},\ \mathsf{mul}(3, X11, X).$

↓ *rewritten by rule* (3.1)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{factorial}(1, X12)},\ \mathsf{mul}(2, X12, X11),\ \mathsf{mul}(3, X11, X).$

↓ *rewritten by rule* (3.1)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{factorial}(0, X13)},\ \mathsf{mul}(1, X13, X12),\ \mathsf{mul}(2, X12, X11),\ \mathsf{mul}(3, X11, X).$

↓ *rewritten by rule* (3.2)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{mul}(1, 1, X12)},\ \mathsf{mul}(2, X12, X11),\ \mathsf{mul}(3, X11, X).$

↓ *rewritten by rule* (3.3)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{mul}(2, 1, X11)},\ \mathsf{mul}(3, X11, X).$

↓ *rewritten by rule* (3.3)

$\mathsf{ans}(X) \leftarrow \underline{\mathsf{mul}(3, 2, X)}.$

↓ *rewritten by rule* (3.3)

$\mathsf{ans}(6) \leftarrow .$

ボディが空の確定節 (**単位節**) $(\mathsf{ans}(6) \leftarrow)$ が得られたとき計算は終了する．単位節は質問の答えをあらわしている (この場合は 3! = 6).

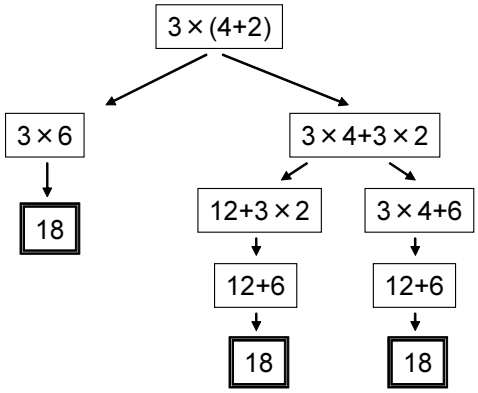


図 3.1　書き換え規則の非決定性

SLD 等の導出原理による計算方法はコンテクストに応じた細かい制御ができないため必然的にバックトラックを生じる．ET ルールによる計算ではバックトラックは発生しない．これは ET ルールの条件部により，節のコンテクストに適した ET ルールが選択されるからである．このように ET ルールによる書き換えはコンテクストに応じた制御が可能で，メモリや計算量において効率のよい計算を実現できる．

なお，ET ルールにはバックトラックの代わりに複数の可能性へ分岐するマルチボディルールがあるが，これについては本論文では扱わないので説明を省略する．

3.2.4　非決定性

適用可能な ET ルールが複数あることがある．そのようなとき，非決定的にひとつのルールが選択され適用される．また節の適用可能な部分の選び方も非決定的である．非決定性は書き換え規則にとって自然な性質である．これを理解するため，節ではなく四則演算を用いた数式「$3\times(4+2)$」を書き換え規則で計算することを例に説明する (図 3.1)．この計算に必要な規則として次が定義されているとする．

規則 1:	$(4+2)$	==>	6
規則 2:	$3\times(4+2)$	==>	$3\times4+3\times2$
規則 3:	3×4	==>	12
規則 4:	3×2	==>	6
規則 5:	3×6	==>	18
規則 6:	$12+6$	==>	18

図 3.1 に示すように規則の適用の仕方によって計算パスは複数存在する．例えば “$3\times(4+2)$” に適用可能な規則と部分は次の二通りある．

- 規則 1 が部分 “$(4+2)$” に対して適用可能．書き換え結果は “3×6”.
- 規則 2 が部分 “$3\times(4+2)$” に対して適用可能．書き換え結果は “$3\times4+3\times2$”.

この例題の書き換え規則は，適用可能ないかなる規則も全て正しい計算結果を導き出すことができる．ET ルールによって確定節を書き換える場合も同様である．一つの確定節に適用可能なルールは複数存在しうる．同じ確定節から到達可能な終了状態も複数ありうる．しかし，その全ての計算パスと全ての終了状態は正しいことが等価変換理論によって保証される．ET ルールはこの性質を満たすように等価変換理論に基づいて作られる．

3.2.5 状態遷移モデルとしての ET 計算

節を状態と考えると，ET ルールによる計算は状態遷移モデルで捉えることができる [4]．例えば，§3.2.3 の計算例では初期状態が

$$\mathsf{ans}(X) \leftarrow \mathsf{factorial}(3, X).$$

であり，ET ルールを適用する毎に節の形が変化してゆくことが状態遷移だと考えられる．そして単位節

$$\mathsf{ans}(6) \leftarrow .$$

は最終状態である．

ET の計算を状態遷移モデルとして捉えることは，宣言的仕様から回路を生成する上で重要である．本研究は ET の状態遷移と等価な状態遷移を実現することにより回路の正当性を保証している．これについて 5.1 章で説明する．

3.3 ET ルールのシンタックス

一般の ET ルールのシンタックスを下記に示す．

Definition 3.3.1. ET ルールのシンタックス

$$H_1, H_2, \ldots, H_k \ \left[, \{\text{cond}\}\right] \ \texttt{==>} \ \left[\{\text{exec}\}, \right] \ B_1, B_2, \ldots, B_\ell. \quad (k \geq 1,\ \ell \geq 0) \qquad (3.4)$$

ここで {cond} は条件部，{exec} は実行部と呼ばれ，条件部には論理式が，実行部には手続きが記述される．括弧 “ [ ] ” は条件部や実行部が文法的に必須ではないことをあらわしている．H_i $(i = 1, 2, \ldots, k)$ および B_j $(j = 1, 2, \ldots, \ell)$ はアトムである．ボディが空のルール (ℓ=0) は許されるが，ヘッドが空のルールは許されない ($k \geq 1$)．

3.3.1 マルチヘッドルール

式 (3.4) で示されるように，ET ルールのヘッドアトムは 1 個とは限らない．複数のヘッドアトムを持つ ET ルールは**マルチヘッドルール**と呼ばれる．これは確定節のボディアトムを複数個まとめて置き換えることを可能にする．このことを次の確定節を例に説明

する．

$$\mathsf{ans}(E) \leftarrow \mathsf{add}(2,B,C),\ \mathsf{sub}(C,2,5),\ \mathsf{mul}(B,C,E).$$

この節に次のマルチヘッド ET ルールを適用する．

$$\mathsf{add}(X,Y,Z),\ \mathsf{sub}(Z,X,W)\ \texttt{==>}\ \mathsf{add}(X,W,Z),\ \mathsf{equal}(Y,W).$$

その結果，節は次のように書き換えられる．

$$\mathsf{ans}(E) \leftarrow \underline{\mathsf{add}(2,5,C),\ \mathsf{equal}(B,5)},\ \mathsf{mul}(B,C,E).$$

本研究では ET ルールをプログラム変換によりマルチヘッドルールへ変形してゆき，最終的にデジタル回路の状態遷移をあらわすマルチヘッドルールを生成する．

3.4 ET ルールのセマンティクス

数学的な議論のため，確定節に ET ルールを適用する数学的な表現について説明しておく．厳密な説明は文献 [6] に譲り，ここでは簡単な具体例を用いて説明した後，一般化された表現について説明する．

3.4.1 具体例による説明

ET ルールが確定節を書き換えるプロセスを具体例を用いて説明する．下記は，この説明に用いる ET ルールである．

$$\mathsf{fact}(N,Z),\{N>0\}\ \texttt{==>}\ \{N1 := N-1\},\ \mathsf{fact}(N1,Z1),\ \mathsf{mul}(N,Z1,Z). \tag{3.5}$$

また，下記はこの説明に用いる確定節 (ans 節) である．

$$\mathsf{ans}(X) \leftarrow \mathsf{fact}(4,X1),\ \mathsf{mul}(5,X1,X). \tag{3.6}$$

上記 ET ルール (3.5) を上記 ans 節 (3.6) に適用する流れは次のようになる．

A. まず最初にルールが ans 節に適用可能かどうかチェックする．
B. 次に，そのルールを用いて ans 節を書き換える

A. 適用可能性のチェック

ルールが適用可能であるためには

1. **[マッチング条件]** ルールヘッドが ans 節のボディにマッチングすること．
2. **[条件部の成立]** ans 節にルールヘッドがマッチしたとき条件部が成立すること．

の二つの条件が満たされればよい．

1. 最初にルールヘッドが ans 節のボディにマッチングすることをチェックする．この例題ではルールに $N = 4$ および $Z = X1$ という代入をすればルールヘッドが ans 節のボディの $\mathsf{fact}(4, X1)$ にマッチすることがわかる．したがってマッチングの条件は成立する．
2. 次に，条件部が成立するかどうかチェックする．先ほどのマッチング代入は下記の集合であらわされる．

$$\theta = \{\, N/4,\ \ Z/X1,\ \ N1/N11,\ \ Z1/Z11 \,\}$$

集合の各要素 "V_i/t_i" は変数 V_i に項 t_i を代入することを表している．最後のふたつの代入 "$N1/N11$" と "$Z1/Z11$" は**変数名変更代入**と呼ばれ，ルール中の変数名が ans 節中の変数名と衝突しないようユニークな名前にリネームしているだけであり，ルールの作用には影響しない．θ をルール (3.5) に適用すると下記のルールが得られる．

$$\mathsf{fact}(4, X1), \{4 > 0\} \ \texttt{==>}\ \{N11 := 4 - 1\},\ \mathsf{fact}(N11, Z11),\ \mathsf{mul}(4, Z11, X1). \quad (3.7)$$

このルール (3.7) の条件部 $\{4 > 0\}$ は真である．

以上の結果から，ルール (3.5) が ans 節 (3.6) に適用可能であることが分かる．

B. 節の書き換え

適用可能なルールを用いて確定節を書き換える手順は以下のようになる．

1. **[節のアトムの置換]** 確定節のボディアトムの置換
2. **[節に対する代入]** ルールの実行部により発生した代入を確定節全体へ適用

この説明には ans 節 (3.6) とルール (3.7) を用いる．

1. まず ans 節 (3.6) のボディのうち，ルール (3.7) のヘッドとマッチした部分が同ルールのボディに置き換えられる (下線部は置き換えられた部分)．

$$\mathsf{ans}(X) \leftarrow \underline{\mathsf{fact}(N11, Z11),\ \mathsf{mul}(4, Z11, X1)},\ \mathsf{mul}(5, X1, X).$$

2. 次にルール (3.7) の実行部 $\{N11 := 4 - 1\}$ が実行され下記の代入が発生する．

$$\rho \ = \ \{\, N11/3 \,\}$$

この代入は ans 節全体に適用され，最終的に次の節が得られる．

$$\mathsf{ans}(X) \leftarrow \mathsf{fact}(3, Z11),\ \mathsf{mul}(4, Z11, X1),\ \mathsf{mul}(5, X1, X). \quad (3.8)$$

3.4.2 ルール適用の一般化された表現

前の例題を引用しながら，一般的な確定節に対する一般的な ET ルールのセマンティクス (ルールの適用の仕方) の表現方法を説明する．これは一般的な ET ルールに対して正当性を議論するのに必要である．

ルールと確定節を集合を用いて表現する

ET ルールのシンタックスは式 (3.4) で示したとおりであるが，数学的な議論を行うには集合を用いた表現が便利である．すなわち，$\mathbb{X}$ を空でないアトム集合，$\mathbb{Y}$ をアトム集合とすると ET ルールは次のようにあらわせる．

$$\mathbb{X}, \{\text{cond}\} \Longrightarrow \{\text{exec}\}, \mathbb{Y}. \tag{3.9}$$

もし条件部が無いルールの場合には cond は常に真であり，もし実行部が無いルールの場合には exec は何もしないもの (nop) とする．

同様に，確定節にも集合を用いた表現を導入する．H を 1 個のアトム，$\mathbb{B}$ をアトムの集合とするとき，任意の確定節は次のように表せる．

$$H \leftarrow \mathbb{B}. \tag{3.10}$$

ルール適用の一般化された表現

■適用条件のチェック　まず最初に適用条件を考える．ET ルール (3.9) が確定節 (3.10) に適用可能であるためには §3.4.1 で示したように

1. **[マッチング条件]** ルールヘッドが確定節のボディにマッチングすること．
2. **[条件部の成立]** 確定節にルールのヘッドがマッチしたとき条件部が成立すること．

の二つの条件が満たされればよい．これは下記を満たす代入 θ が存在することと同じである．

$$(\mathbb{X}\theta \subseteq \mathbb{B}) \wedge (\text{cond}\theta = \text{true}) \tag{3.11}$$

条件式の $(\mathbb{X}\theta \subseteq \mathbb{B})$ はルールヘッドが確定節のボディにマッチングすることをあらわし，$(\text{cond}\theta = \text{true})$ はルールの条件部が満たされることをあらわしている．

■確定節の書き換え　続いて，確定節をルールによって書き換える．もし，条件 (3.11) を満たすルールが複数存在する場合には，その中からただ一つのルールを**非決定的**に選択する．書き換えの手順は §3.4.1 で示したように以下のようになる．

1. **[節のアトムの置換]** 確定節のボディアトムをルールボディに置き換える．
2. **[節に対する代入]** ルールの実行部により発生した代入 ρ_{exec} を節全体へ適用する．

書き換えに用いる ET ルールはマッチングのための θ が施された次のルールを用いる．

$$\mathbb{X}\theta, \{\text{cond}\}\theta \Longrightarrow \{\text{exec}\}\theta, \mathbb{Y}\theta. \tag{3.12}$$

条件 (3.11) を満たす θ が存在するとき，確定節 (3.9) のボディ $\mathbb{B}$ を，ルールヘッドにマッチする部分 $\mathbb{X}\theta$ と，それ以外の部分 $\mathbb{B}'$ に分けることができる．したがって，確定節は次

のようにあらわせる．

$$H \leftarrow \mathbb{X}\theta \cup \mathbb{B}'.$$
$$\text{where } \mathbb{B}' = \mathbb{B} \setminus \mathbb{X}\theta$$

確定節は，まずボディアトムが置き換えられる．

$$H \leftarrow \underline{\mathbb{X}\theta} \cup \mathbb{B}'.$$
$$\Downarrow$$
$$H \leftarrow \underline{\mathbb{Y}\theta} \cup \mathbb{B}'.$$

続いて execθ の実行によって節全体に変数への代入 ρ_{exec} が発生する．その結果，次式で表される節が得られる．

$$\left\{ H \leftarrow \mathbb{Y}\theta \cup \mathbb{B}' \right\}\rho_{\text{exec}} = \left\{ H\rho_{\text{exec}} \leftarrow \mathbb{Y}\theta\rho_{\text{exec}} \cup \mathbb{B}'\rho_{\text{exec}} \right\} \tag{3.13}$$

例題との対応

下記は，§3.4.1 の具体的な例題 (ルール (3.5), 確定節 (3.6)) と一般化された表現 (ルール (3.9), 確定節 (3.10)) の対応関係を示したものである．

$$\begin{aligned} H &= \mathsf{ans}(X) \\ \mathbb{B} &= \{\, \mathsf{fact}(4, X1),\ \mathsf{mul}(5, X1, X) \,\} \\ \mathbb{X} &= \{\, \mathsf{fact}(N, Z) \,\} \\ \mathbb{Y} &= \{\, \mathsf{fact}(N1, Z1),\ \mathsf{mul}(N, Z1, Z) \,\} \\ \text{cond} &= (N > 0) \\ \text{exec} &= \{N1 := N - 1\} \end{aligned}$$

例題のルールが確定節に適用可能であることは，適用条件 (3.11) を満たす θ が存在することから示される．

$$\theta = \{\, N/4,\ Z/X1,\ N1/N11,\ Z1/Z11 \,\}$$

この θ が適用条件 (3.11) を満たすことは次のように確認できる．

$$\begin{aligned} \mathbb{X}\theta &= \{\, \mathsf{fact}(4, X1) \,\} \subseteq \mathbb{B} \\ \text{cond}\theta &= (4 > 0) = \text{true} \end{aligned}$$

したがって例題のルールは確定節に適用可能である．

節にルールを適用する時，execθ の実行により代入 $\rho_{\text{exec}} = \{\, N11/3 \,\}$ が発生し，下記の式が成り立つ．

$$\begin{aligned} H\rho_{\text{exec}} &= \mathsf{ans}(X) \\ \mathbb{Y}\theta\rho_{\text{exec}} &= \{\, \mathsf{fact}(3, Z11),\ \mathsf{mul}(4, Z11, X1) \,\} \\ \mathbb{B}'\rho_{\text{exec}} &= \{\, \mathsf{mul}(5, X1, X) \,\} \end{aligned}$$

式 (3.13) に上記の結果を代入すると，確かに式 (3.8) が得られることがわかる．

3.5 ET の特徴

- ET パラダイムでは確定節によって仕様を宣言的に定義する．確定節で定義することにより宣言的仕様から曖昧さを排除できる．また，宣言的仕様には問題の性質や範囲のみを記述しアルゴリズムは含まない．アルゴリズムは宣言的仕様を満たすものの中から任意に選択し，その手続きを ET ルールで記述する．宣言的仕様を変更することなくアルゴリズムを選択できることは SLD に対する大きなアドバンテージである．
- 宣言的仕様とアルゴリズムの間には**等価変換理論**により正当性が保証される．ET では宣言的仕様とアルゴリズムが分離されており，宣言的仕様は確定節で記述され，手続きは ET ルールで記述される．それらの関係を結ぶ理論が等価変換理論であり意味の保存という原理に基づいている．
- ET ルールは節に対する書き換え規則である．ET ルールで節を書き換えることは等価変換である．
- ET ルールはコンテクストに応じた制御が可能であり効率の良い計算が可能．
- ET ルールはそれ自体をプログラム変換できる．ET ルールはアルゴリズムを記述するが，そのレベルは宣言的仕様に近い抽象的なレベルから具体的なメモリアクセスやレジスタ転送レベル (RTL) まで幅広く記述できる．そして等価変換理論と 5 章で説明する「代数系変換理論」によって正当性を保ちながら ET ルールを様々なレベルへプログラム変換することができる．これにより，様々な計算プラットフォームで等価な計算が可能である．
- ET ルールを宣言的仕様から自動生成する研究が進められており [7–12]，ET ルールを自動生成する理論が作られている．その研究と本研究によって宣言的仕様から計算システムや回路を自動生成できるようになる．
- ET ルールは非決定的に適用される．ある質問節に対して適用可能な ET ルールが複数ある場合には，その中からひとつのルールが非決定的に選択され適用される．したがって同じ質問に対して計算パスが一通りとは限らない．
- ET ルールはマルチヘッドが扱える．

3.5.1 正当性

ET を使う最も大きな理由はしっかりとした正当性保証の枠組みがあることである．一般のプログラム理論によれば，プログラムの正当性を示すためには「部分正当性」と「停止性」の二つが示されればよい．しかし従来のプログラミング言語では部分正当性を示すことは容易ではない．それらの言語で部分正当性を示す場合，事前条件・事後条件・不変式を見つけ出し数学的帰納法によって証明するという手法が用いられる．この方法ではプログラムがある程度完成した後でないと部分正当性の検証はできないし，プログラムに変

更が生じると正当性も変わってしまう．

等価変換理論では背景知識 $\mathbb{D}$ と質問節 $s_0 \in \mathbb{Q}$ に対して意味がフォーマルに定義される．これを $\mathcal{M}(\mathbb{D} \cup \{s_0\})$ と表す．背景知識 $\mathbb{D}$ は知識節の集合である．ET 計算モデルにおける部分正当性は，s_k と s_{k+1}(s_k を書き換えルール $\boldsymbol{r}_k$ で書き換えた節) の間に $\mathcal{M}(\mathbb{D} \cup \{s_k\}) = \mathcal{M}(\mathbb{D} \cup \{s_{k+1}\})$ の関係が成り立てばよい．書き換えルールは互いに依存しないので，個々の書き換えルール $\boldsymbol{r}_k$ が独立に上記を満たしていればよい．ルールの追加や削除によって他のルールの正当性が変わることはない．従って ET ルール (正しいルール) を追加しながらプログラムを構築すれば正しいプログラムを得ることができる．この性質を利用して Component–wise な開発が可能である [52]．また正当性を保ったままプログラムを変更することも容易である．

3.5.2 プログラム変換

本研究の特徴は ET ルールをプログラム変換して回路を生成していることである．ET ルールはプログラム変換しやすい．そしてプログラム変換の正当性は，等価変換理論を用いて保証することができる．本研究が ET ルールを用いる最大の理由はこの性質にある．

第 4 章

ET ルールを用いた回路生成手法

この章では，本研究の実験的な手法であり基礎となった「ET ルールから」デジタル回路を生成する基本的なアイデア [53] についての説明を行う．この章で説明するアイデアは非常に単純で実用的なものではない．しかし，その内容は 5 章で考察され，抽象化されて汎用の理論へ発展してゆく．

4.1 デジタル回路で実現可能なクラス

まず最初に，任意の ET ルールの記述が回路化できる訳ではないことを説明しておく．一般にデジタル回路は記憶素子として有限個のフリップフロップしか内蔵していないため，扱える状態数は有限である．つまりデジタル回路で実現可能なのは**有限状態マシン** [13–15] である[*1]．一方，ET ルールは一般再帰が記述でき，データとしてリスト構造も扱えるため有限状態マシン以上の記述能力を持っている[*2]．

デジタル回路で実現できないルール記述への対応としてソフトウェアとの協調計算への発展も考えられる．ソフトウェアによる柔軟さと回路による並列計算を併用することで，効率のよい計算を実現する方法である．これについては，本書の後半で協調計算システム生成のフレームワーク (6 章) や実験的な協調計算システムの構築例 (7 章) を紹介している．

4.1.1 デジタル回路による有限状態マシン

図 4.1 にデジタル回路による有限状態マシンの簡単な構成図を示す．その動作は次のようになっている．

[*1] CPU でさえ単体では有限状態マシンでしかない．(無限の) 外部記憶を接続しなければチューリングマシン [54] と等価な計算可能性を持つことはできない．

[*2] ET ルールはチューリング完全な記述能力を持つと考えられる．チューリング完全な言語である Constraint Handling Rules (CHR) から ET ルールへ変換する研究が存在する [55].

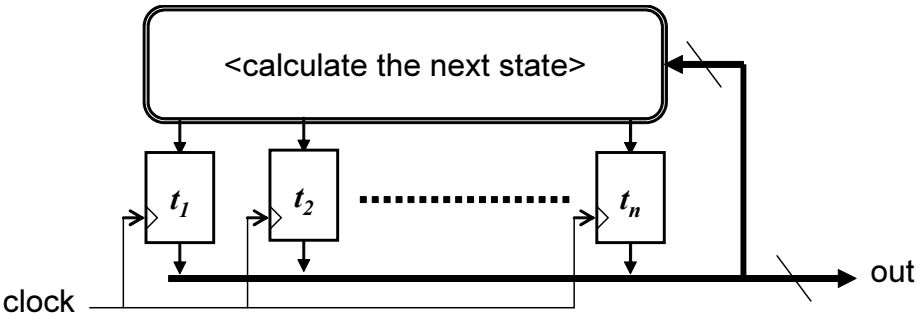


図 4.1　Finite State Machine Circuit

- 状態 $\langle t_1, t_2, \ldots, t_n \rangle$ は**フリップフロップ** (§1.3.3) によって保持されている．
- "<calculate the next state>" の部分は**遷移関数**と呼ばれ,「現在の状態」から「次の状態」を計算し，それをフリップフロップに渡す役目をする．
- クロック信号 (clock) は，フリップフロップが「次の状態」を取り込むタイミングを与える．

遷移関数はクロックの合図が来る前に「次の状態」の計算を終えなければならない．この条件を満たすため，遷移関数は**組み合わせ回路** (§1.3.4) だけで構成される．

4.1.2　有限状態マシンの ET ルール

デジタル回路で実現できるものは有限状態マシンなので，回路化可能なルールも有限状態マシンで実行可能なクラスに限定される．与えられた ET ルールが回路化可能かどうかの議論は大きな研究テーマとなるため，代わりに図 4.1 の有限状態マシンで実行可能なもっとも単純な ET ルールの形を示し，そのルールの形式に還元できるものが回路化可能であると考える．その単純なルールの形を以下に示す．

$$\begin{aligned} & p_1(t_1), p_2(t_2), \ldots, p_n(t_n) \\ & \quad \texttt{==>}\ \{ < \text{calculate the next state} > \}, \\ & \quad\quad p_1(s_1), p_2(s_2), \ldots, p_n(s_n). \end{aligned} \tag{4.1}$$

ルール (4.1) のヘッドを「現在の状態」，ボディを「次の状態」，実行部を「遷移関数」とみなし，ルールの適用はクロック (clock) の合図で行われるとする．このアナロジーが図 4.1 の回路に適合するには，ルール (4.1) に対して下記の制約も必要になる．

1. ルールヘッドとルールボディは引数だけが異なる．
2. アトムの引数に可変リストを持たない (整数へコーディング可能なものだけ).
3. 条件部を持たない (条件不成立のときは同じ状態へ遷移するよう変換).
4. 実行部は**組み合わせ回路**で実現可能な記述だけが許される．
5. ヘッドの変数 (t_i) への代入が発生しない (変数代入は引数の置換に変換).

4.1.3 回路生成の着想

ET ルールは**プログラム変換**が容易な言語である．もし与えられた ET ルールを前述の制約を満たす (4.1) の形にプログラム変換できれば回路化は容易である．この考え方が本研究の基礎となっている．

以降で簡単な例題を用いて，ET ルールを上記の制約を満たすルールへプログラム変換する基本的なアイデアを説明する．

4.2 階乗の例題を用いた回路生成の説明

確定節を「状態」として捉えると，アトムの引数は記憶領域 (レジスタ，F/F) であり，アトムは単に記憶領域をグループ化したものにすぎない．計算過程で確定節が変化してゆくことは状態遷移であり [4]，回路のレジスタが更新されてゆくことに対応する．状態を書き換える ET ルールからは遷移関数を得ることができる．

ここでは具体的に ET ルールの記述から回路を生成する過程を例題で示す．簡単な例題として factorial を計算する回路を作る．

4.2.1 等価変換計算モデルによる階乗の計算例

回路を実現するには計算アルゴリズムが必要である．アルゴリズムは ET プログラム (=ET ルールの集合) によって与える．ここでは下記の ET プログラムを用いる．ただしこのプログラムの正しさは分かっているものとする．

$$\begin{aligned} &\mathsf{factloop}(N, M, F), \{N \neq 0\} \\ &\quad ==> \{K := N - 1;\ L := N \times M;\}, \\ &\qquad \mathsf{factloop}(K, L, F). \end{aligned} \tag{4.2}$$

$$\mathsf{factloop}(N, M, F), \{N = 0\} ==> \{F := M;\}. \tag{4.3}$$

ルール集合 {(4.2),(4.3)} が factorial を計算する ET プログラムである．F, K, L, M, N は変数で := は右辺の計算結果を左辺の変数へ代入することを表している．この ET プログラムは末尾再帰で factorial を定義したものであり，X=$n!$ は $\mathsf{factloop}(n, 1, X)$ と等価である ($\mathsf{factloop}(N, M, F)$ の M は累積 (積算) 変数の役割をしている)．

このルールを使った factorial の計算を以下に例示する．まず，ET ルールを用いて計算するには最初に質問の節 (ans 節) を与える必要がある．節とは論理式である．この例題で用いる ans 節は下記の形をしている．

$$\mathsf{ans}(X) \leftarrow \mathsf{factloop}(n, 1, X). \tag{4.4}$$

式 (4.4) は ← の右辺が真のとき左辺が真であることを示す．n には非負整数を与える．具体例として 3 の階乗を計算する質問節 (ans 節) を下記に示す．

$$\mathsf{ans}(X) \leftarrow \mathsf{factloop}(3, 1, X).$$

ET 計算モデルでは，節を ET ルールで書き換えてゆくことが “計算” である．例題の二つの ET ルールのうち，この節に適用可能なのはルール (4.2) だけである．なぜなら factloop の第一引数が 3 なのでルール (4.2) の条件部は成り立つがルール (4.3) の条件部は成り立たないからである．ルール (4.2) によって ans 節は下記のように書き換えられる．

$$\begin{aligned} \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(3, 1, X). \\ &\Downarrow \boxed{\text{apply rule (4.2)}} \\ \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(2, 3, X). \end{aligned}$$

得られた ans 節に適用可能なルールは今度もルール (4.2) だけである．このように，ans 節にルール (4.2) が次々に適用され続け，次の形の ans 節が得られたところでルール (4.2) が適用できなくなる．

$$\mathsf{ans}(X) \leftarrow \mathsf{factloop}(0, 6, X).$$

しかし今度はルール (4.3) が適用可能である．

$$\begin{aligned} \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(0, 6, X). \\ &\Downarrow \boxed{\text{apply rule (4.3)}} \\ \mathsf{ans}(6) &\leftarrow . \end{aligned}$$

最後にボディが空の節が得られる．これは ans(6) が無条件に真であることをあらわし，質問「3 の階乗は X である」を満たす X の存在が “証明” されたことを示している．

4.2.2 ルール，節と回路の関係

節とルールが回路とどのように関係するのかについて論じる．まずルールの役割を調べるため先ほどの節の変化を示す．

$$\begin{aligned} \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(3, 1, X). \\ &\Downarrow \\ \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(2, 3, X). \\ &\Downarrow \\ \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(1, 6, X). \\ &\Downarrow \\ \mathsf{ans}(X) &\leftarrow \mathsf{factloop}(0, 6, X). \\ &\Downarrow \\ \mathsf{ans}(6) &\leftarrow . \end{aligned}$$

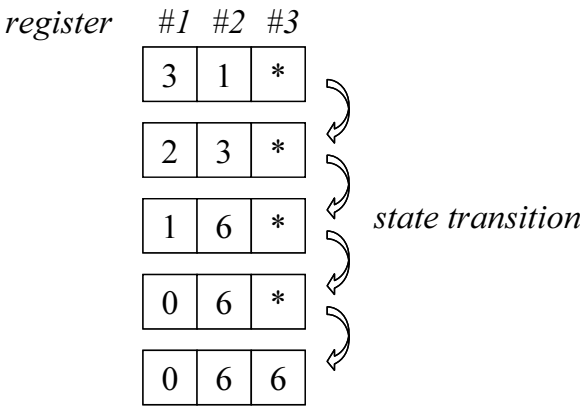


図 4.2　State Transition of Factorial

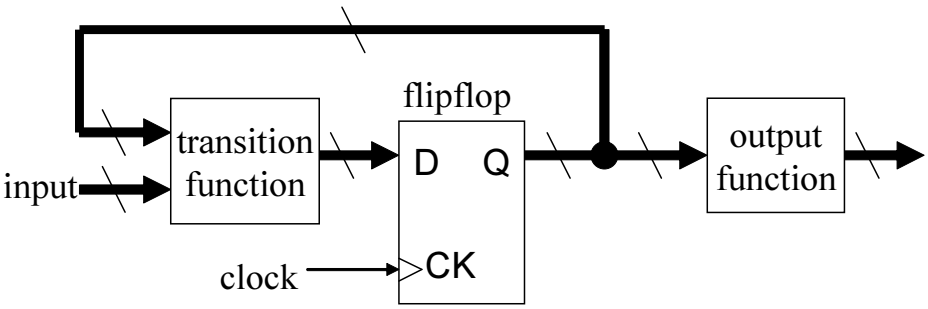


図 4.3　Moore Finite State Machine

ルールを適用する毎に factloop の引数が変化する．つまりルールは状態を遷移させる役割をする．この状態遷移をレジスタを用いて簡潔に表現すると図 4.2 となる．図中の *register #1, #2, #3* は factloop の 3 つの引数に相当する．

続いて節の役割を調べる．ans 節 (4.4) は計算の初期状態である．図 4.2 では最上部の枠で表されるレジスタの状態がそれにあたる．また ans アトムの引数 X は結果が返る場所を表している．つまり ans 節は計算の初期状態を与え，結果が返る場所を指定する．これは回路の入出力インタフェースに相当する．

図 4.2 の各状態は上の ET プログラム {(4.2),(4.3)} による計算の各状態へマッピングすることができる．つまり 図 4.2 と同じ状態遷移を行う回路はその動作を ET プログラム {(4.2),(4.3)} の動作にマッピングできるので答えの正当性が保証される．そのような回路を実現するために本手法では 図 4.3 で示す Moore FSM (Finite State Machine) [14] を採用した回路の実現方法を示す[*3]．Moore FSM は一般的に広く用いられている状態遷移マシンであり 3 つの要素から構成される．フリップフロップ (flipflop) は現在の状態を保持

[*3] Mealy FSM[15] でない理由は，本手法の計算結果が図 4.3 に示すように内部状態そのものであり，出力関数に入力信号 (input) を必要としないからである．

する．遷移関数 (transition function) は現在の状態と現在の入力から次の状態を計算する．出力関数 (output function) は現在の状態から値を計算し出力する．これら 3 つの要素のうち遷移関数と出力関数を適切に決めることによって所望の状態遷移マシンを作ることができる．

以降では ET ルールから図 4.3 の遷移関数と出力関数を求める方法を説明する．

4.2.3 ルールを回路へ変換する基本戦略

この章では ET ルールを Moore FSM に適合させるために変更を加える．実用的な回路にするためには計算終了を知らせる仕組みと計算結果を保持する仕組みが必要である．そのために ET ルールに終了フラグの追加と，計算終了時に終了状態を保持するルールを追加する．また，遷移関数と出力関数の求め方についても説明する．

終了フラグと終了状態保持ルールの追加

ルール集合 {(4.2), (4.3)} のアトムの引数に終了フラグを追加し，終了時にだけ適用される新ルールを追加する．この結果，以下のルール集合 {(4.5), (4.6), (4.7)} が得られる．

$$\begin{aligned}&\mathsf{factloop}(\mathbf{0}, N, M, F), \{N \neq 0\}\\ &\quad ==> \{K := N-1;\ L := N \times M;\},\\ &\qquad \mathsf{factloop}(\mathbf{0}, K, L, F).\end{aligned} \tag{4.5}$$

$$\begin{aligned}&\mathsf{factloop}(\mathbf{0}, N, M, F), \{N = 0\}\\ &\quad ==> \{F := M;\},\\ &\qquad \mathbf{factloop}(\mathbf{1}, N, M, F).\end{aligned} \tag{4.6}$$

$$\begin{aligned}&\mathbf{factloop}(\mathbf{1}, N, M, F)\\ &\quad ==>\ .\end{aligned} \tag{4.7}$$

ルール (4.5), (4.6) はルール (4.2), (4.3) を次のように変換したものである．

- それぞれのルールの factloop アトム（ヘッドアトムと再帰アトム）の引数の先頭にフラグ **0** を追加する．
- 停止ルール (4.3) のボディの最後にヘッドアトムのフラグを **1** にしたもの

  $$\mathbf{factloop}(\mathbf{1}, N, M, F)$$

  を追加する．ルールは末尾再帰ルールになる（ルール (4.6)）．

ルール (4.7) は新たな停止ルールである．計算が終了したときにだけ適用される．これら新たなプログラム {(4.5), (4.6), (4.7)} に対する ans 節は次のようになる．

$$\mathsf{ans}(X) \leftarrow \mathsf{factloop}(\mathbf{0}, n, 1, X). \tag{4.8}$$

この新たな ans 節をプログラム {(4.5), (4.6), (4.7)} で計算することは，式 (4.4) で示した ans 節 をプログラム {(4.2), (4.3)} で計算するのと全く等価であることが証明できる[*4].

続いて，回路が計算結果を保持するためにルール (4.7) を次のように変更する．

$$\begin{aligned} &\mathsf{factloop}(1, N, M, F) \\ &\quad ==> \ \mathbf{factloop}(\mathbf{1}, \boldsymbol{N}, \boldsymbol{M}, \boldsymbol{F}) \ . \end{aligned} \tag{4.9}$$

この変更によって回路は計算終了の状態を永久に保持できるようになるが，プログラムとしては停止性を満たさなくなる．しかし，このことは正当性に影響しない．このことについては 5 章で議論する．

ルールから状態関数を求める

ここでは input 信号を除いた回路を考える．input は初期値をロードするだけであり計算中は状態遷移が内部状態だけで決まるためである．input は後から追加することができる．

末尾再帰ルール (4.5), (4.6), (4.9) はそれぞれ特定の条件下での状態遷移 (state transition) を表している．例えば質問 $\mathsf{ans}(X) \leftarrow \mathsf{factloop}(0, 3, 1, X)$ にルール (4.5) を適用する計算過程を見てみると

$$\begin{array}{c} \mathsf{ans}(X) \leftarrow \mathsf{factloop}(0, 3, 1, X). \\ \Downarrow \quad \boxed{\text{apply rule (4.5)}} \\ \mathsf{ans}(X) \leftarrow \mathsf{factloop}(0, 2, 3, X). \end{array}$$

ルール (4.5) を質問に適用した結果 factloop アトムの引数の値のみが変化している．つまりルール (4.5) は 図 4.4 のような遷移関数回路を表していると考えられる．遷移関数はルールの実行部によって表現され，入力はヘッドアトム，出力はボディアトムが表現している．図 4.4 の遷移関数回路は $(Flag{=}0) \wedge (N{\neq}0)$ の場合のみ適用できる．それはヘッドのパターンマッチングと条件部の成立という要件に由来する．同様に残りの 2 つのルール (4.5),(4.9) からもそれぞれの条件下での遷移関数が得られる．

条件別に得られた 3 つの遷移関数を一つにまとめる．そのためにはヘッドのパターンマッチングおよび条件部によってルールが選択される機構に着目する．つまり，各々のルールの出力をそのルールの適用条件の評価結果によってスイッチするような機構を実現すればよい (図 4.5)．各ルールの実行部は次状態の候補を計算する回路になる．その候補の中から適切なものがスイッチによって選ばれる．そのスイッチを制御するのがルールの適用条件である．これを一般化したものが 図 4.6 である．

ルールから出力関数を求める

変換されたルール {(4.5), (4.6), (4.7)} による計算結果は，計算終了フラグが 1 になったときの factloop アトムの第三引数 X の値である．そこで，終了フラグが 1 になったら変

[*4] 証明は省略するが終了フラグを除いて状態遷移が同じであることから理解できる．

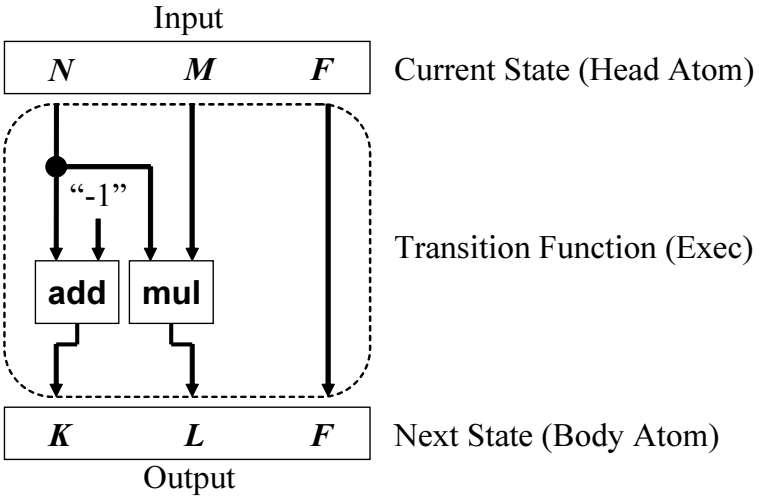
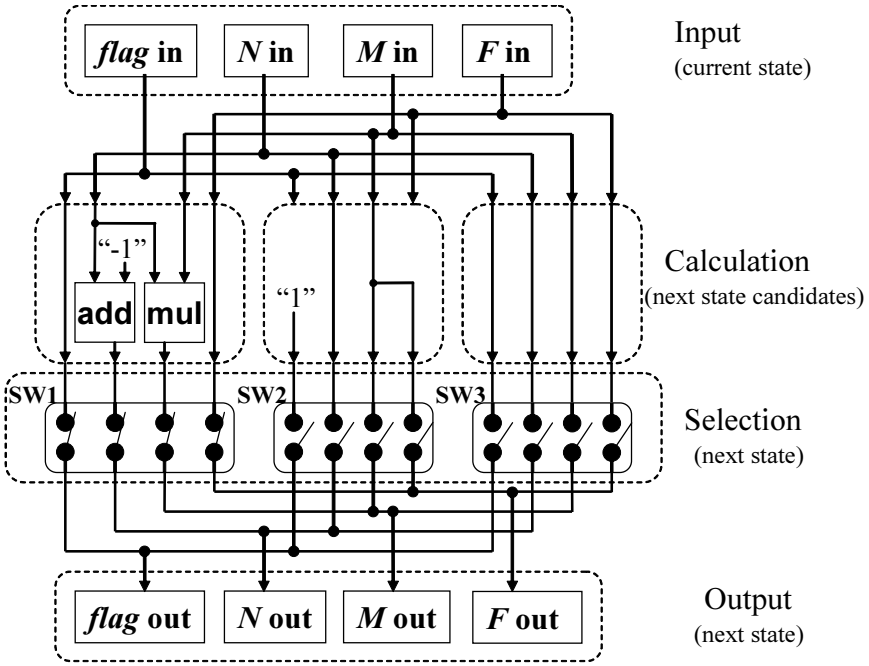


図 4.4　Transition Circuit Block Diagram corresponding to ET Rule (4.5)



SW1 closes iff $(N \neq 0) \wedge (flag = 0)$, SW2 closes iff $(N = 0) \wedge (flag = 0)$, SW3 closes iff $(flag = 1)$.

図 4.5　Factorial Transition Function Circuit

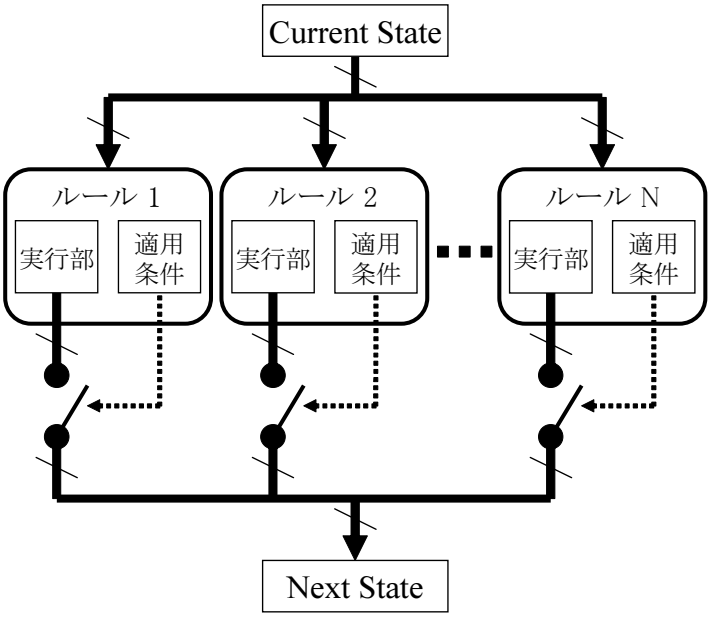


図 4.6　General Transition Function Circuit

数 X の内容を出力すればよい．また，終了フラグも一緒に出力すれば，計算結果を取り出すタイミングも分かるようになる．

Input Circuit and Computation Invoking

ここではユーザーが計算開始の指示を発行できるよう新たな制御信号 – *WRITE* 信号 – を導入して計算開始の機構を回路に追加する．この modification は FSM 回路でしばしば問題となる未定義状態からの脱出にも貢献する．

計算の開始は ans 節を与えることに相当する．ET プログラムの計算の開始は 式 (4.8) 形式の 質問を与えることで起動される．回路における計算の開始は (4.8) 式の初期状態をレジスタにセットすること，すなわちユーザーデータ n と初期値 $flag$=0, M=1 をフリップフロップへロードすることで開始される．

ユーザーデータと初期値をロードする目的で *WRITE* 信号を導入する．*WRITE* 信号は，それがアサートされている間は遷移関数の出力に優先してユーザーデータと初期値がフリップフロップへロードされる．この機構によりユーザーデータは *WRITE* 信号がアサートされている期間の最後のクロックの立ち上がりエッジで有効な値が取り込まれる．そして *WRITE* 信号がデアサートされてから自動的に計算が開始される．計算が終了すると次のユーザーデータがロードされるまで計算結果を保持している．

この機構は *WRITE* 信号がアサートされると終了フラグを 0 にする．そのため回路の状態によらず新たな計算が開始されるという特徴を持つ．従ってリセットが必要なく，FSM が未定義状態に入っても *WRITE* 信号のアサートによって正常な動作に戻ることができる．

4.3 Synthesis of Factorial Circuit

基本的なアイデアは以上に述べた．しかし実際の回路を生成するには ET のルールをもう少し回路記述へ近づけた方が生成しやすい．そこでルールを ET の空間で変換して，より低位な記述にする．

4.3.1 Merging of ET Rules

ルール {(4.5), (4.6), (4.9)} をマージして等価な 1 本のルールにする．それには適用条件（ヘッドのパターンマッチングと条件部）の評価結果によって実行部の代入式を選択するようにすればよい．

$$
\begin{aligned}
&\mathsf{factloop}(I, N, M, F)\\
&\quad ==> \{\ B1 := (I == 0)\ \ \mathtt{and}\ \ (N \neq 0)\,;\\
&\qquad B2 := (I == 0)\ \ \mathtt{and}\ \ (N == 0)\,;\\
&\qquad J := \mathtt{if}\ \ \{B1\}\ \ \mathtt{then}\ \ 0\\
&\qquad\qquad \mathtt{else\ if}\ \ \{B2\}\ \ \mathtt{then}\ \ 1\\
&\qquad\qquad \mathtt{else}\ \ 1;\\
&\qquad K := \mathtt{if}\ \ \{B1\}\ \ \mathtt{then}\ \ (N-1)\\
&\qquad\qquad \mathtt{else\ if}\ \ \{B2\}\ \ \mathtt{then}\ \ N\\
&\qquad\qquad \mathtt{else}\ \ N;\\
&\qquad L := \mathtt{if}\ \ \{B1\}\ \ \mathtt{then}\ \ (N \times M)\\
&\qquad\qquad \mathtt{else\ if}\ \ \{B2\}\ \ \mathtt{then}\ \ M\\
&\qquad\qquad \mathtt{else}\ \ M;\\
&\qquad G := \mathtt{if}\ \ \{B1\}\ \ \mathtt{then}\ \ F\\
&\qquad\qquad \mathtt{else\ if}\ \ \{B2\}\ \ \mathtt{then}\ \ M\\
&\qquad\qquad \mathtt{else}\ \ F;\\
&\quad \},\\
&\quad \mathsf{factloop}(J, K, L, G).
\end{aligned}
\tag{4.10}
$$

実行部の $B1, B2$ はそれぞれルール (4.5), (4.6) の適用条件の評価値である．評価値によって次状態 J, K, L, G の計算式をセレクタで切り替えている[*5]．ルール (4.9) は他のルールが適用できない場合のデフォルトなので適用条件の評価値を計算する必要はない．

ルール (4.10) は，この 1 本で {(4.5), (4.6), (4.9)} と全く等価なプログラムとなっている．つまりこのルールの実行部は完全な遷移関数を表しているのである．

[*5] $(\mathtt{if}\ \{c\}\ \mathtt{then}\ e_A\ \mathtt{else}\ e_B)$ という式は，c が 1 (true) ならば e_A の値を返し，0 (false) ならば e_B の値を返す式である．

第 5 章

回路生成法の理論的考察と拡張

前章では ET ルールから回路を生成する基本的な考え方を説明した．この章では，回路の正当性を数学的に議論し，その議論を足掛かりに回路生成手法を拡張する．この手法は回路生成のみならず，ET ルールからプログラムの実行バイナリを生成する方法 (すなわち ET ルールのコンパイラ) にも応用可能なものである．

4 章で紹介した回路生成方法は等価なプログラムへの変換を繰り返して実現されている．その手法はアドホックに開発されたものであり，理論的な考察に基づいたものではなかった．ここで 4 章の回路生成の流れを一般化してみると，図 5.1 のような枠組みに当てはめることができる．*S* は与えらた宣言的仕様である．*S* に対して正しい ET プログラムは複数存在しうる．その中から *P* という ET プログラムを選び，それを回路化しやすいよう変換する．そのような変換方法も複数存在しうる．変換されたルールの中から *R* を選び，それを回路に変換する．*R* から回路を生成する方法も幾つか存在しうる．

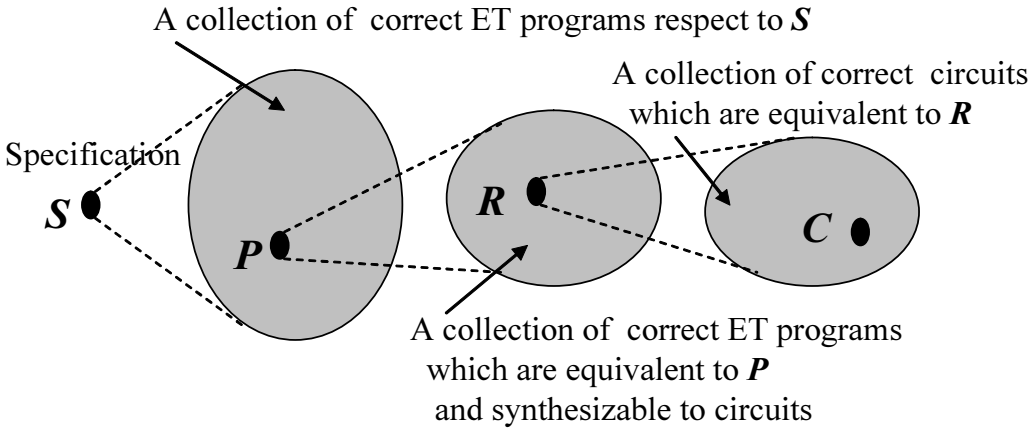


図 5.1　Circuit Synthesis Scheme

4 章で述べた研究成果は上記の枠組みのうち，P から R への変換方法を一つと R から C への変換方法を一つ発見したにすぎない．デジタル回路が宣言的仕様に対して正当であるとはどういうことか,「確定節」というドメインと「レジスタ」というドメインの間の正当性関係をどのように考えればよいのか，それらを理論的に考察することによって回路生成手法の拡張が可能になる．

5.1 回路の正当性

本書で扱う正当な回路とは，宣言的仕様を満たすものを言う．したがって次のように回路の正当性を定義する．

Definition 5.1.1. 回路の正当性の定義
宣言的仕様に対して正当な回路とは，宣言的仕様の質問に対する正しい答えを見つけることができる回路である．

以下では宣言的仕様に対して正当な回路がどのようなものかを理解するため，例題を用いて説明する．

5.1.1 正当な回路の例題

例題の回路の宣言的仕様

Example 5.1.1. 例題の回路の宣言的仕様

$$\begin{array}{rcl} \mathbb{D} & = & \{\, \mathsf{sub}(n,m,x) \leftarrow . \mid n,m,x \in \mathbb{Z},\ x = n-m \,\} \\ \mathbb{Q} & = & \{\, \mathsf{ans}(X) \leftarrow \mathsf{sub}(n,m,X). \mid n,m \in \mathbb{Z},\ X \in \mathbb{V} \,\} \end{array} \tag{5.1}$$

ここで $\mathbb{Z}, \mathbb{V}$ はそれぞれ整数の集合, 変数の集合である．上記の例では $\mathbb{D}$ が確定節の無限集合となっているが，通常 sub/3 のような基本的な述語は**ビルトイン述語**として用意されており，明確な解釈が与えられている．したがって，その述語の定義を記述しなくても誤解なく使うことができる（この例題では sub/3 の解釈を読者に明確に伝えるため便宜上記述している）．この宣言的仕様が定義するものは引き算の答えを問い合わせる問題である．次の質問節を例に用いて説明する．

例題の質問節

$$\mathsf{ans}(X) \leftarrow \mathsf{sub}(5,2,X). \tag{5.2}$$

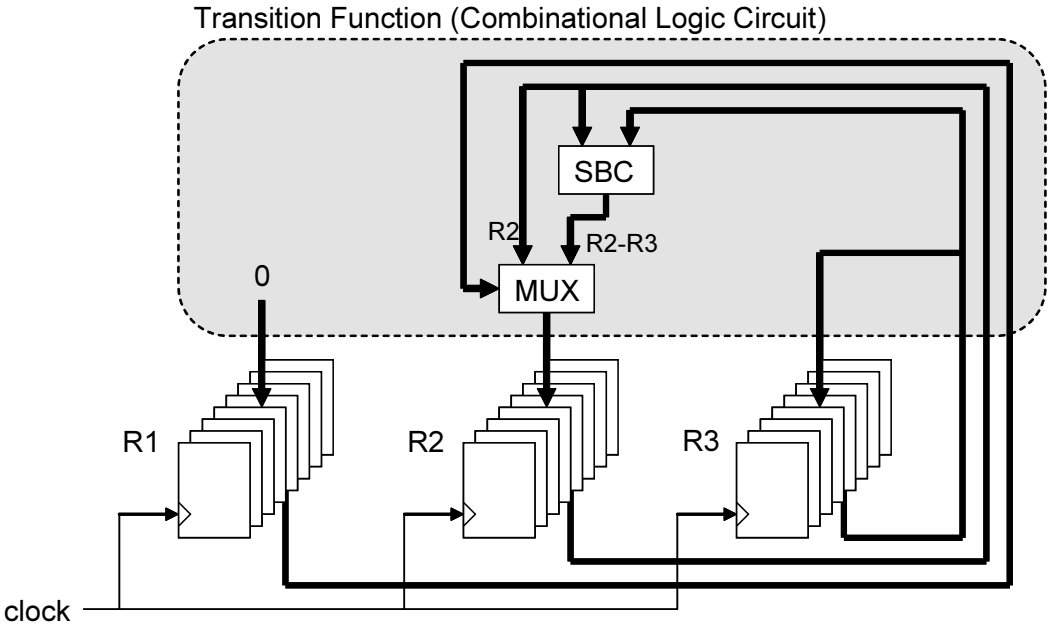


図 5.2　例題の宣言的仕様に対して正当な計算をする回路の一例

例題の正当な答え

例題の正しい答えは，背景知識 $\mathbb{D}$ と確定節 (5.2) の論理的帰結である．この例題の場合には，次に示すように節空間 S_{cl} 上で知識 $\mathbb{D}$ を用いて簡単に演繹できる．

$$\mathsf{ans}(X) \leftarrow \mathsf{sub}(5, 2, X).$$
$$\downarrow$$
$$\mathsf{ans}(3) \leftarrow .$$

最終的に得られる ans(3) が答えである．

例題の宣言的仕様に正当な回路

例題の宣言的仕様は引き算の答えを求めるだけなので簡単な回路で実現できる．実現方法はいくつか考えられるが，ここでは 3 つのレジスタ R_1, R_2, R_3 で計算を行う回路を考える (図 5.2)．レジスタはフリップフロップ (§1.3.3) のアレイ (配列) によって実現される．遷移関数は，減算回路 (SBC)，セレクタ (MUX) および定数回路 (0) から構成され，これらすべてが組み合わせ回路 (§1.3.4) だけで作られている．

レジスタの状態を $[R_1, R_2, R_3]$ と表すことにする．この回路の動作は，次の状態空間 $\mathbb{Z}^3$ とレジスタ状態を遷移させる遷移関数 $\mathbb{R}_{\mathrm{cir}}$ で表すことができる．

宣言的仕様 (5.1) に正当な回路の動作

$$\begin{aligned} \mathbb{Z}^3 &= \left\{ [R_1,\ R_2,\ R_3] \mid R_1, R_2, R_3 \in \mathbb{Z} \right\} \\ \mathbb{R}_{\mathrm{cir}} &= \left\{ \begin{array}{lll} [1,\ R_2,\ R_3] & \mapsto & [0,\ R_2 - R_3,\ R_3] \\ [0,\ R_2,\ R_3] & \mapsto & [0,\ R_2,\ R_3] \end{array} \right\} \end{aligned} \tag{5.3}$$

R_1 はフラグの働きをしており，$R_1 = 1$ のとき計算中を，$R_1 = 0$ のとき計算終了を表す．遷移関数 $\mathbb{R}_{\mathrm{cir}}$ は書き換え規則の形で表している．記号 "$\mapsto$" の左辺が現在の状態，右辺が次の状態を表している．遷移関数は $R_1 = 1$ のとき R_2 と R_3 の差を計算した結果を R_2 へ代入する．そして計算が終了したことを表すため R_1 に 0 を代入する．$R_1 = 0$ のときは現状態を保持する．この回路に質問 (5.2) の計算を実行させるには，レジスタに下記の初期値を与える．

回路の初期状態

$$[1, 5, 2] \tag{5.4}$$

回路は状態空間 $\mathbb{Z}^3$ 上でクロックに同期して自律的に次の状態遷移を行う．

$$\begin{array}{c} [1, 5, 2] \\ \big\downarrow \text{transition by } \mathbb{R}_{\mathrm{cir}} \\ [0, 3, 2] \end{array}$$

最終状態として $[0, 3, 2]$ が得られる．得られた状態の R_1 の値は計算終了を示し，R_2 の値は質問の答え $(5 - 2 = 3)$ を示している．

5.1.2 数学的な定義

前章の例題から，ET と回路が同じ結果を導いていることを**感覚的**に理解できる．しかし厳密には ET の結果は節であり回路の結果は整数の列なので数学的に回路の正当性を保証していない．また，定義 Definition 5.1.1 は正当な回路の満たすべき性質を自然言語で記述したものである．回路生成の理論を構築するには，この定義を数学的に定義しなおす必要がある．

以下では定義 Definition 5.1.1 を数学的な言葉で表すための議論を行う．

ET ルールによる計算を足がかりにする

次に，宣言的仕様に対して正当な下記の ET ルールで計算してみる．

宣言的仕様 (5.1) に正当な ET ルール

$$\mathbb{R}_{\mathrm{ET}} = \Big\{ \mathsf{sub}(N, M, Z), \{num(N), num(M), var(Z)\} \texttt{ ==> } \{Z := N - M;\}. \Big\} \quad (5.5)$$

$num(\alpha)$ は α が数ならば真を返し，そうでなければ偽を返すビルトインの論理関数である．$var(\beta)$ は β が変数ならば真を返すビルトイン関数である．(5.5) のルールが意味を保存する (つまり等価変換である) ことの証明は省く．ET ルールの生成方法の説明は他の文献 [7–12] に譲り，ここでは上記 ET ルールが与えられているものとして話を進める．

ET ルール (5.5) に質問節 (5.2) を与えると質問節は状態空間 $\mathbb{S}_{\mathrm{cl}}$ 上で次の状態遷移を行う．

$$\begin{array}{l} \mathsf{ans}(X) \leftarrow \mathsf{sub}(5, 2, X). \\ \qquad\qquad \Big\downarrow \text{rewritten by } \mathbb{R}_{\mathrm{ET}} \\ \mathsf{ans}(3) \leftarrow . \end{array}$$

最終的に基礎単位節が得られ計算は終了する．ルール (5.5) が ET ルールなので，この計算結果の正当性は等価変換理論によって保証される．上記の正当な回路が，ET と同じ計算結果を得ていることが確認できる．

5.1.3 ET と回路の状態の比較

ET の計算では節を状態と考えるので状態空間は全ての節の集合 $\mathbb{S}_{\mathrm{cl}}$ である．一方，例題の回路は 3 つのレジスタに整数を保持するので状態空間は $\mathbb{Z}^3$ である．このように ET と回路では状態空間に違いがあり，直接の比較はできない．このような異なる状態空間の間で計算を理論的に比較するには，両者の状態の間になんらかの対応関係を定義する必要がある．

関数 $\mathcal{F}$ および Ψ の導入

Example 5.1.1 に対して，ET の状態空間と回路の状態空間の間の対応関係を定義する下記の関数 $\mathcal{F}, \Psi$ を導入する．

ET の状態空間と回路の状態空間の間の写像

$$\mathcal{F}\Big(\{ \mathsf{ans}(X) \leftarrow \mathsf{sub}(n, m, X). \} \Big) \stackrel{\mathrm{def}}{=} [1, n, m]$$

$$\Psi\ (\ [0, n, m]\ ) \stackrel{\mathrm{def}}{=} \{ \mathsf{ans}(n) \leftarrow . \}$$

$\mathcal{F}$ は節から回路の状態への部分写像であり，ET の質問節を回路の初期状態へ写す．Ψ は回路の状態から節への部分写像であり，回路の終了状態を ET の終了状態に写す．例題の

質問節 (5.2) は写像 $\mathcal{F}$ によって次のように回路の状態 (5.4) へ写される．

$$\mathcal{F}\Big(\{\, \mathsf{ans}(X) \leftarrow \mathsf{sub}(5,2,X). \} \Big) \;=\; [1,5,2]$$

これを回路の初期状態とし，回路上で計算すると回路は終了状態 $[0,3,2]$ で停止する．この回路の終了状態を写像 Ψ によって節へ写像すると，それは ET による終了状態 (つまり答え) になっている．

$$\Psi\,(\,[0,3,2]\,) \;=\; \{\, \mathsf{ans}(3) \leftarrow . \,\}$$

この写像 $\mathcal{F}, \Psi$ によって，任意の質問節 $s_0 \in \mathbb{Q}$ に対しても回路の計算結果と ET の計算結果がマッチすることは容易に確認できる．

5.1.4 回路の正当性を数学的に再定義する

Example 5.1.1 に限らず，一般の問題に対してもこのような $\mathcal{F}, \Psi$ を適切に定義することで，ET の計算と回路の計算とを比較できる．ここでは，これを足がかりに回路の正当性を議論し，数学的な表現で回路の正当性を定義しなおす．

ET による正当性の原理

最初に，ET がどのような仕組みで正当性を保証しているかを説明する．ET の計算は $s_0 \in \mathbb{Q}$ を初期値とし，ET ルール集合 $\mathbb{R}_{\mathrm{ET}}$ の中から適用可能なルールを非決定的に選択して書き換えを行う．今仮に次の図ような計算パスをたどるとする．

$$\begin{array}{l} s_0 \\ \big\downarrow r_1 \\ s_1 \\ \big\downarrow r_2 \\ \vdots \\ \big\downarrow r_n \\ s_n \end{array}$$

この図の s_i は節であり r_i は適用された ET ルールを表している．各 r_i は宣言的仕様に対して正しい ET ルールであり意味を保存する (§2.4.2)．すなわち任意の i について下記が成り立つ．

$$\mathcal{M}(s_{i-1}) = \mathcal{M}(s_i)$$

したがって初期状態 s_0 と停止状態 s_n の間に下記が成り立つ．

$$\mathcal{M}(s_0) = \mathcal{M}(s_n)$$

もし s_n が終了状態 (すなわちグラウンドな単位節) ならばそれが s_0 の答えである．つまり下記を満たす s_n を発見することが正しい計算となる．

ET の正当性保証の原理

$$\mathcal{M}(s_0) = \mathcal{M}(s_n) \quad s_n \in \boldsymbol{Fin} \tag{5.6}$$

ここで $\boldsymbol{Fin}$ は終了状態の集合，すなわち基礎アトム (p.19) からなる単位節の集合である．正しく作られた ET ルール集合 $\mathbb{R}_{\mathrm{ET}}$ は常にこの条件を満たす．

回路の正当性の考え方

ET の正当性の原理は，式 (5.6) で示されるように，質問の節と答えの節の間で “意味” が保存されることである．回路の初期状態と終了状態を節に対応づけることで，回路にも同じ原理が適用できる．すなわち，質問節 s_0 を写像 $\mathcal{F}$ で回路上の状態 m_0 に写し，それを回路で計算し，回路上で得られた結果 m_k を写像 Ψ で節 s_n に写したとき，$\mathcal{M}(s_0)$=$\mathcal{M}(s_n)$ が成り立てばよい．これを図示すると次のようになる．

$$\begin{array}{ccccc} & \overset{\mathcal{F}}{} & & & \\ s_0 & \longmapsto & m_0 & & (\, m_0 = \mathcal{F}(s_0)\,) \\ & & \big\downarrow r'_1 & & \\ & & m_1 & & \\ & & \big\downarrow r'_2 & & \\ & & \vdots & & \\ & & \big\downarrow r'_k & & \\ s_n & \longmapsto & m_k & & (\, s_n = \Psi(m_k)\,) \\ & \underset{\Psi}{} & & & \end{array}$$

m_j $(j = 0, 1, \ldots, k)$ は回路の状態でありレジスタ上の数値列で表現される．r'_j はレジスタ上の数値列を遷移させる規則である．数値列はクロックに同期して $m_0 \mapsto m_1 \mapsto \cdots$ と次々に遷移してゆき最終状態 m_k に到達する．このような m_j の状態遷移列を実現する回路を作成すれば，その回路を使って s_0 の答えである s_n を見つけることができる．

なお，一般に ET 上の遷移数 n と回路上の遷移数 k は一致する必要はない $(n \neq k)$．定義 Definition 5.1.1 を満たすには式 (5.6) が成り立てばよく，遷移数は定義と無関係だからである[*1]．また，4 章で紹介した回路生成手法において，ルール (4.9) の形が停止性を満たしていないことを指摘したが (p.48)，上記の議論から回路の正当性を侵害していないことが分かる (状態は終了状態から遷移していない)．

[*1] 本書の正当性の定義には時刻の概念を含まない．時刻の概念を含むケースは今後の課題である (§8.1.2)

回路の正当性の数学的定義

以上の議論を踏まえて，正当な回路の数学的定義は次のようになる．

Definition 5.1.2. 回路の正当性の数学的定義宣言的仕様 $\langle \mathbb{ID}, \mathbb{Q} \rangle$ に対して正当である回路とは次を満たす回路である．

> $\mathbb{Q}$ の任意の要素を回路の初期状態に変換する写像 $\mathcal{F} : \mathbb{Q} \to \mathbb{S}'$ が定義され，回路の最終状態を $\mathbb{S}_{\mathrm{cl}}$ の元へ写す部分写像 $\Psi : \mathbb{S}' \to \mathbb{S}_{\mathrm{cl}}$ が定義され，それらが任意の $s_0 \in \mathbb{Q}$ に対して次を満たす:
>
> $$\text{“回路に } m_0{=}\mathcal{F}(s_0) \text{ を与えると } m_k \text{ を返す”}$$
> $$\Rightarrow \text{ “}\mathcal{M}(\mathbb{ID} \cup \Psi(m_k)){=}\mathcal{M}(\mathbb{ID} \cup s_0) \text{ かつ } \Psi(m_k) \in \boldsymbol{Fin}\text{”}$$

ただし，$\mathbb{S}_{\mathrm{cl}}$ は節集合 ($\mathbb{S}_{\mathrm{cl}} \supseteq \mathbb{Q}$)，$\mathbb{S}'$ は回路の状態空間，$\boldsymbol{Fin}$ は終了状態の集合 ($\boldsymbol{Fin} \subseteq \mathbb{S}_{\mathrm{cl}}$) である．

回路の実現方法によって状態表現が異なるため $\mathcal{F}, \Psi$ は回路毎に異なる．また，同じ回路であってもこれらの関数が一意に決まるとは限らない．§5.2 では，この数学的定義を足がかりに理論的な展開を行う．

回路の正当性を議論するポイントは，回路上での計算構造と ET 計算モデルの計算構造を比較することである．$\mathcal{F}$ および Ψ を導入するのは，これら異なる世界の物事を単一の世界にそろえ，その単一世界の言葉で代数学的構造の比較をするためである．

5.2 理論的考察

ET ルールから回路を生成することは，**ET 計算モデルという代数的構造**から**回路という代数的構造**への変換として捉えることができる．ET ルールによる計算を回路上の計算に変換するための数学的な基礎理論が必要である．

この章では代数系変換理論，ルールのマージ理論，ルールから FSM への変換理論を数学的に議論する．

5.2.1 代数系変換

ET ルールから正しい回路を生成するために必要な代数系変換について議論する．

代数系

一般に，状態の集合 $\mathbb{S}$ と $\mathbb{S}$ 上の演算の集合 $\mathbb{R}$ の組

$$\langle \mathbb{S}, \mathbb{R} \rangle$$

を代数系と呼ぶ．ET の場合の状態は確定節であり，演算は書き換え規則なので

$$\mathbb{S} = \mathbb{S}_{\mathrm{cl}}$$
$$\mathbb{R} = \mathbb{R}_{\mathrm{ET}}$$

である．また，§5.1.1 の回路では

$$\mathbb{S} = \mathbb{Z}^3$$
$$\mathbb{R} = \mathbb{R}_{\mathrm{cir}}$$

である．FSM の状態遷移関数は書き換え規則と見ることができる．例えば §5.1.1 の回路の $\mathbb{R}_{\mathrm{cir}}$ は状態 $[1, R_2, R_3]$ を $[0, R_2 - R_3, R_3]$ に遷移させるので次の書き換え規則として表せる．

$$\mathbb{R}_{\mathrm{cir}} = \left\{ [1, R_2, R_3] \Longrightarrow [0, R_2 - R_3, R_3] \right\}$$

ただし ET ルールとは異なりルール適用は決定的である．§5.1.2 の例でみた ET の計算が回路の計算と一致することは，ET の代数系

$$\langle \mathbb{S}_{\mathrm{cl}}, \mathbb{R}_{\mathrm{ET}} \rangle$$

の計算が，状態マシン回路の代数系

$$\langle \mathbb{Z}^3, \mathbb{R}_{\mathrm{cir}} \rangle$$

でも実行可能であることを意味する．そして，§5.1.3 で見たように ET の代数系 $\langle \mathbb{S}_{\mathrm{cl}}, \mathbb{R}_{\mathrm{ET}} \rangle$ と回路の代数系 $\langle \mathbb{Z}^3, \mathbb{R}_{\mathrm{cir}} \rangle$ は関数 $\mathcal{F}$ と Ψ で結び付けられる (図 5.3).

このように宣言的仕様に対して正当な計算を行う代数系 $\langle \mathbb{S}, \mathbb{R} \rangle$ があるとき，代わりに別の代数系 $\langle \mathbb{S}', \mathbb{R}' \rangle$ 上で正当な計算を行うための条件を以降で議論する．

5.2.2 最も単純な代数系変換

宣言的仕様に対して正当な計算を行う二つの代数系 $\langle \mathbb{S}, \mathbb{R} \rangle$ と $\langle \mathbb{S}', \mathbb{R}' \rangle$ の関係で最も簡単なのは両者の代数的構造が同一の場合である．すなわち $\mathbb{S}$ と $\mathbb{S}'$ の要素の間に一対一対応がつけられ，$\mathbb{S}$ 上の状態遷移と $\mathbb{S}'$ 上の状態遷移が同一にみなせる場合である．

このケースでは $\mathcal{F}$ を $\mathbb{S}$ から $\mathbb{S}'$ への全単射として定義可能である．また Ψ は $\mathcal{F}^{-1}$ として定義できる．そして $\mathcal{F}$ および Ψ によって状態遷移が保存される．すなわちこのケースの条件は次のようになる

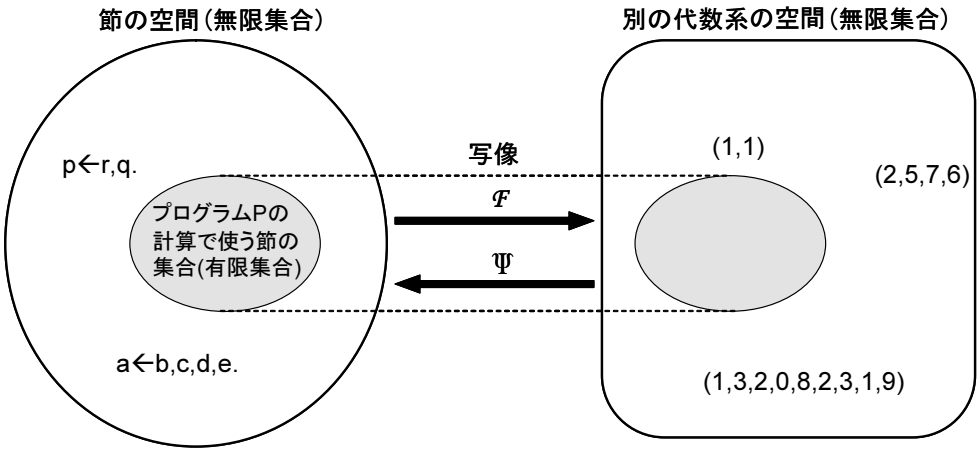


図 5.3　代数系変換

Theorem 5.2.1. 単純な代数系変換
$\forall s_1, s_2 \in \mathbb{S}$ および $\forall m_1, m_2 \in \mathbb{S}'$ に対して下記が成り立つとき，代数系 $\langle \mathbb{S}, \mathbb{R} \rangle$ 上の正当な計算結果は $\langle \mathbb{S}', \mathbb{R}' \rangle$ 上での計算で求めることができる．

・全単射なる $\mathcal{F} : \mathbb{S} \to \mathbb{S}'$ が存在する．(逆写像を Ψ とする．$\Psi(\mathcal{F}(s_1)) = s_1$)　(5.7)

・$\mathbb{R}$ に $s_1 \mapsto s_2$ の遷移ルールが存在
$\Rightarrow$ $\mathbb{R}'$ に $\mathcal{F}(s_1) \mapsto \mathcal{F}(s_2)$ の遷移ルールが存在　(5.8)

・$\mathbb{R}'$ に $m_1 \mapsto m_2$ の遷移ルールが存在
$\Rightarrow$ $\mathbb{R}$ に $\Psi(m_1) \mapsto \Psi(m_2)$ の遷移ルールが存在　(5.9)

このとき正当性は次のように証明できる．

Proof 5.2.1. 正当性の証明
$\forall s_0 \in \mathbb{S}$ に対して $\mathcal{F}(s_0)$ (これを m_0 とする) を $\mathbb{R}'$ で書き換えたときに得られる $\mathbb{S}'$ 上の状態列を考える．

$$m_0,\ m_1,\ \ldots,\ m_k$$

1. このそれぞれの要素に Ψ を適用して得られる下記の列は $\mathbb{S}$ 上の等価変換列である．
$$\Psi(m_0),\ \Psi(m_1),\ \ldots,\ \Psi(m_k)$$
なぜなら m_{i-1} を m_i に書き換えるルールを r'_i とすると，条件 (5.9) により
$$r_i : \Psi(m_{i-1}) \mapsto \Psi(m_i)$$
を満たす ET ルール r_i が $\mathbb{R}$ に存在するので $\mathcal{M}(\Psi(m_{i-1})) = \mathcal{M}(\Psi(m_i))$ が成り立つ (§2.4.2 参照).
2. もし m_k が $\mathbb{S}'$ 上の終了状態ならば $\Psi(m_k)$ は $\mathbb{S}$ 上の終了状態である．なぜなら $\Psi(m_k)$ が終了状態でないとすると $\Psi(m_k)$ から遷移可能な状態 $\alpha \in \mathbb{S}$ が存在する．すると条件 (5.8) より $\mathcal{F}(\Psi(m_k)) = m_k$ から状態 $\mathcal{F}(\alpha)$ へ遷移が可能であり，m_k が終了状態であることに反する．従って $\Psi(m_k)$ は終了状態でなければならない．
3. 以上 1.2. より $\Psi(m_k)$ は質問 s_0 と等価な意味を持つ $\mathbb{S}$ 上の終了状態なので s_0 の答えである．

■

これは回路の正当性の定義 Definition 5.1.2 を満たす．

5.2.3 条件のゆるい代数系変換

ここでは，$\mathcal{F}$ や Ψ が全単射にならないケースを議論する．

上の代数系変換では $\mathcal{F}$ と Ψ が全単射である．しかし §5.1.3 で示した $\mathcal{F}, \Psi$ は全単射ではない．例えば回路上の状態 $[0, n, m]$ を Ψ で写像することを考える．

$$\Psi\left(\ [0, n, m]\ \right) \ = \ \{\, ans\,(n) \leftarrow .\,\}$$

この場合，任意の m に対して同じ値を返すので Ψ は単射ではないことが分かる．一般に $\mathcal{F}$ や Ψ は全単射である必要はない．以下ではもう少し条件を緩めた $\mathcal{F}$ や Ψ の満たすべき性質を議論する．

要素の区別

まず $\mathbb{S}$ 上の二つの異なる要素 a, b は $\mathbb{S}'$ 上へマッピングしても区別できなければならないので $\mathcal{F}$ は単射でなければならない.

$$\mathcal{F} : \mathbb{S} \to \mathbb{S}' \text{ が単射である} \quad i.e. \quad \forall a, b \in \mathbb{S} \left(a \neq b \ \Rightarrow \ \mathcal{F}(a) \neq \mathcal{F}(b) \right)$$

また $\mathbb{S}$ の要素を $\mathcal{F}$ で $\mathbb{S}'$ 上へマッピングしたものは Ψ によって $\mathbb{S}$ 上の同じ要素に戻らなければならないので次の条件が必要である.

$$\forall a \in \mathbb{S} \quad \Psi\left(\mathcal{F}\left(a\right)\right) = a$$

この条件から，任意の $a \in \mathbb{S}$ が Ψ の値域に入ることがわかる．つまり Ψ の値域は $\mathbb{S}$ の上位集合 (Superset) である.

$$\Psi(\mathbb{S}') \supseteq \mathbb{S} \quad i.e. \quad \forall a \in \mathbb{S} \ \exists m \in \mathbb{S}' \left(\Psi(m) = a \right)$$

等価変換の要請

$\mathbb{S}'$ 上の状態遷移が等価変換となるためには $\forall m_1, m_2 \in \mathbb{S}'$ に対して次が成り立つ必要がある.

“$\Psi(m_1) \in \mathbb{S}$ かつ $\mathbb{R}'$ に $m_1 \mapsto m_2$ の遷移ルールが存在する”
$\Rightarrow$ “$\Psi(m_2) \in \mathbb{S}$ かつ $\mathbb{R}$ に $\Psi(m_1) \mapsto \Psi(m_2)$ の遷移ルールが存在する”

計算の停止性を保証するために次の条件が必要である.

“$\Psi(m_1) \in \mathbb{S}$ かつ $\mathbb{R}'$ に $m_1 \mapsto m_2$ の遷移ルールが存在しない”
$\Rightarrow$ “$\mathbb{R}$ に $\Psi(m_1) \mapsto \Psi(m_2)$ の遷移ルールが存在しない”

以上の議論からゆるい代数系変換の条件は次のようにまとめられる.

Theorem 5.2.2. 条件のゆるい代数系変換
$\forall s_1, s_2 \in \mathbb{S}$ および $\forall m_1, m_2 \in \mathbb{S}'$ に対して下記が成り立つとき，代数系 $\langle \mathbb{S}, \mathbb{R} \rangle$ 上の正当な計算結果は $\langle \mathbb{S}', \mathbb{R}' \rangle$ 上での計算で求めることができる．

$$\text{単射な } \mathcal{F} : \mathbb{S} \to \mathbb{S}' \text{ が存在する} \quad i.e. \quad \forall s_1, s_2 \in \mathbb{S} \left(s_1 \neq s_2 \ \Rightarrow \ \mathcal{F}(s_1) \neq \mathcal{F}(s_2) \right) \tag{5.10}$$

$$\mathcal{F} \text{ に対して } \Psi \circ \mathcal{F} \text{ が恒等写像となるような } \Psi \text{ が存在する } \left(\Psi\left(\mathbb{S}'\right) \supseteq \mathbb{S} \right) \tag{5.11}$$

$$\begin{aligned} &\text{“}\Psi(m_1) \in \mathbb{S} \text{ かつ } \mathbb{R}' \text{ に } m_1 \mapsto m_2 \text{ の遷移ルールが存在する”} \\ &\ \Rightarrow \ \text{“}\Psi(m_2) \in \mathbb{S} \text{ かつ } \mathbb{R} \text{ に } \Psi(m_1) \mapsto \Psi(m_2) \text{ の遷移ルールが存在する”} \end{aligned} \tag{5.12}$$

$$\begin{aligned} &\text{“}\Psi(m_1) \in \mathbb{S} \text{ かつ } \mathbb{R}' \text{ に } m_1 \mapsto m_2 \text{ の遷移ルールが存在しない”} \\ &\ \Rightarrow \ \text{“}\mathbb{R} \text{ に } \Psi(m_1) \mapsto \Psi(m_2) \text{ の遷移ルールが存在しない”} \end{aligned} \tag{5.13}$$

この正当性を以下に示す．

Proof 5.2.2. 正当性の証明
$\forall s_0 \in \mathbb{S}$ に対して $m_0 = \mathcal{F}(s_0)$ を $\mathbb{R}'$ で書き換えたときに得られる $\mathbb{S}'$ 上の状態列を考える．

$$m_0,\ m_1,\ \ldots,\ m_k$$

1. このそれぞれの要素に Ψ を適用して得られる下記の列は $\mathbb{S}$ 上の等価変換列である．
$$\Psi(m_0),\ \Psi(m_1),\ \ldots,\ \Psi(m_k)$$
なぜなら条件 (5.12) により m_{i-1} から m_i へ遷移可能で $\Psi(m_{i-1}) \in \mathbb{S}$ ならば，$\Psi(m_i) \in \mathbb{S}$ であり $r_i : \Psi(m_{i-1}) \mapsto \Psi(m_i)$ なる ET ルール $r_i \in \mathbb{R}$ が存在する．従って $\mathcal{M}(\Psi(m_{i-1})) = \mathcal{M}(\Psi(m_i))$ が成り立つ．今 $\Psi(m_0) = s_0 \in \mathbb{S}$ なので全ての i について $\Psi(m_i) \in \mathbb{S}$ かつ $\mathcal{M}(\Psi(m_{i-1})) = \mathcal{M}(\Psi(m_i))$ が成り立つ．
2. もし m_k が $\mathbb{S}'$ 上の終了状態ならば $\Psi(m_k)$ は $\mathbb{S}$ 上の終了状態である．なぜなら m_k が終了状態ならばいかなる $\beta \in \mathbb{S}'$ に対しても m_k から β へ遷移するルールは存在しない．すると条件 (5.13) より，$\Psi(m_k)$ から $\Psi(\beta)$ に遷移する ET ルールは存在しない．関数 Ψ の値域は $\mathbb{S}$ の superset なのでいかなる $s \in \mathbb{S}$ に対しても $\Psi(m_k)$ から s へ遷移する ET ルールは存在しない．よって $\Psi(m_k)$ は終了状態である．
3. 以上 1.2. より $\Psi(m_k)$ は s_0 と等価な意味を持つ $\mathbb{S}$ 上の終了状態なので s_0 の答えである．

■

これは回路の正当性の定義 Definition 5.1.2 を満たす．

もし Ψ が全射，すなわち $\Psi(\mathbb{S}') = \mathbb{S}$ のときには，$\Psi(m_1) \in \mathbb{S}$ と $\Psi(m_2) \in \mathbb{S}$ は自明であり，条件 (5.12) と条件 (5.13) の代わりに次の式を用いることができる．

$$\forall m_1, m_2 \in \mathbb{S}' \Big(\text{“}\mathbb{R}' \text{ に } m_1 \mapsto m_2 \text{ の遷移ルールが存在する”} \\ \Leftrightarrow \text{ “}\mathbb{R} \text{ に } \Psi(m_1) \mapsto \Psi(m_2) \text{ の遷移ルールが存在する”}\Big)$$

この式は条件 (5.12) と条件 (5.13) の対偶から導かれる．

5.2.4 一般化

もう一度ドメイン変換の条件を下記示す.

$$\mathcal{F} : \mathbb{S} \to \mathbb{S}' \text{ が単射である } i.e.\ \forall a, b \in \mathbb{S} \left(a \neq b \ \Rightarrow\ \mathcal{F}(a) \neq \mathcal{F}(b) \right) \tag{5.14}$$

$$\forall a \in \mathbb{S}\ \ \Psi(\mathcal{F}(a)) = a \tag{5.15}$$

$$\Psi(\mathbb{S}') \supseteq \mathbb{S} \quad i.e.\ \forall a \in \mathbb{S}\ \ \exists m \in \mathbb{S}' \left(\Psi(m) = a \right) \tag{5.16}$$

$$\forall m, n \in \mathbb{S}' \Big(\text{“}\mathbb{R}' \text{ に } m \mapsto n \text{ の遷移ルールが存在する”} \\ \Leftrightarrow \text{ “}\mathbb{R} \text{ に } \Psi(m) \mapsto \Psi(n) \text{ の遷移ルールが存在する”} \Big) \tag{5.17}$$

ただし Ψ を全射であると仮定してしてる．もし Ψ が全射でないなら，Ψ が全射となるように $\mathbb{S}'$ の範囲を限定すればよい．この操作を行っても $\mathcal{F}$ の単射の条件は保存される．

以前の議論では $\mathbb{S}$ を節集合，$\mathbb{S}'$ を数値列の集合として考えていたが，この条件式には節や数値に特化したものが含まれていない．一般の代数系 $\langle \mathbb{S}, \mathbb{R} \rangle$ と $\langle \mathbb{S}', \mathbb{R}' \rangle$ について適用できる．すると次のような応用が可能である．

■プログラム変換　$\mathbb{S}$ と $\mathbb{S}'$ 両方とも確定節集合とし，$\mathbb{R}$ と $\mathbb{R}'$ が ET ルールであるとすれば，ET のプログラム変換の正当性が議論できる．

■ET コンパイラ　$\mathbb{S}$ を節集合，$\mathbb{S}'$ をメモリ上の数値とすれば ET コンパイラの正当性が議論できる．

5.3 状態空間の決定

冗長性のない回路を実現するには適切な $\mathcal{F}$ および Ψ を定義する必要がある．このため**状態空間**を明らかにする．ここでの状態空間とは，宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$ で定義された質問 $\mathbb{Q}$ の各要素から ET ルール $\mathbb{R}$ によって到達可能なすべての状態を集めた集合を指す．

状態空間を調べる手法として**メタ計算**を用いる．もし状態空間が有限でない場合にはメタ計算が停止しない可能性がある．しかし，デジタル回路で実現可能ものは有限状態マシンであり，回路化可能なルールは有限状態マシンのルールに還元できるため，状態空間は有限であることが保証される．

なお，本研究の手法をプログラムのコンパイラに応用する場合には必ずしもメタ計算は必要ない．むしろ，プログラムは状態空間が無限になるケースを扱うことができるのでメタ計算が停止しない可能性が高い．たとえ状態空間が不明でも，スタックやポインタを用いれば任意の確定節をメモリ上のデータとして表現でき[*2]，それらに対する $\mathcal{F}, \Psi$ を定義することは可能である．

[*2] もちろん，コンパイラへ応用する場合でも状態空間を明らかにする意義はある．実行効率のよいバイナリや，スタック / ヒープのオーバーフローの危険が無いバイナリを生成できる可能性がある．

```
// r1:
  (p *x *y) ==> (g *x *m), (h1 *m 0 *y).
// r2:
  (g *x *z) ==> (h2 *x 0 *z).
// r31:
  (h1 *x *s *w), {(number *x), (> *x 0)}
      ==> {(:= *x1 (- *x 1)), (:= *s1 (+ *s *x))}, (h1 *x1 *s1 *w).
// r32:
  (h2 *x *s *w), {(number *x), (> *x 0)}
      ==> {(:= *x1 (- *x 1)), (:= *s1 (+ *s *x))}, (h2 *x1 *s1 *w).
// r41:
  (h1 0 *s *w) ==> {(= *s *w)}.
// r42:
  (h2 0 *s *w) ==> {(= *s *w)}.
```

図 5.4　メタ計算の例題に用いるルール

以降の説明では，メタ計算するルールが回路化可能なルールであると仮定する (したがってメタ計算は停止する)．なお，メタ計算の手法については現在研究途中であり体系化されていない．ここで紹介する方法は実験レベルのものであることを断っておく．

5.3.1　例

ここではルール $\mathbb{R}$ として図 5.4 を用いる．これは実際の ET インタプリタ [20] 上での実験に用いた記述である．ET インタプリタが扱う記述は DEC–10 PROLOG とシンタックスが異なる．アトムは S 式として記述され，括弧の最初のシンボルが述語名をあらわし，スペースで区切られて引数が続く．アスタリスク (*) から始まるシンボルは変数である．

質問節として次の節を用いる．

(ans *y) ← (p n *y).　　n は非負整数

例えば図 5.5 は，質問節の n が 2 の場合の実行例である．図 5.5 の実行例の中に出現する節は，次の 5 種類に分類できる．

```
 0  (ans *A) <- (p <int> *A).
 1  (ans *A) <- (g <int> *B), (h1 *B <int> *A).
2~4 (ans *A) <- (h1 *B <int> *A), (h2 <int> <int> *B).
5~8 (ans *A) <- (h1 <int> <int> *A).
 9  (ans <int>) <- .
```

```
step 0: (ans *A) <- (p 2 *A)
step 1: (ans *A) <-          (g 2 *B),(h1 *B 0 *A).
step 2: (ans *A) <-                   (h1 *B 0 *A),(h2 2 0 *B).
step 3: (ans *A) <-                   (h1 *B 0 *A),(h2 1 2 *B).
step 4: (ans *A) <-                   (h1 *B 0 *A),(h2 0 3 *B).
step 5: (ans *A) <-                   (h1  3 0 *A).
step 6: (ans *A) <-                   (h1  2 3 *A).
step 7: (ans *A) <-                   (h1  1 5 *A).
step 8: (ans *A) <-                   (h1  0 6 *A).
step 9: (ans  6) <-.
```

図 5.5　例題のルールによる実行例

この 5 種類の節を有限個のレジスタを用いて表現できれば，次の質問に対する回路を生成できる．

$$\text{(ans *y)} \leftarrow \text{(p 2 *y)}.$$

回路が $\mathbb{Q}$ の任意の質問に対して計算できるには，少なくとも $\mathbb{Q}$ のすべての要素に対する実行例を含む有限の状態空間を調べる必要がある．

ここでは，メタ計算の結果が有限の空間であれば $\mathbb{Q}$ の実行例以外を含んでいても構わないものとする．すなわち，この章のメタ計算で得られる空間は状態空間の上位集合 (superset) である．

5.3.2　メタ計算

図 5.6 はメタ計算の結果を示している．各状態の左肩の数字は状態の種類を示している．同じ数字の状態は同じ種類の状態である．

まず番号 0 のメタ節を説明する．これは全ての質問を表現する ans 節である．

```
(ans <var *Z>) <- (p <int s1> <var *Z>)
```

このメタ節のボディアトム p の第一引数は整数 (<int>)，第二引数は変数 (<var>) である．メタ節のデータは次の形式で与えられている．

< データ型　ラベル >

同一の ans 節の中で同じラベルを持つデータは同じ値を持っている．特にデータ型が var (変数) で同じラベルを持つものはリンクしている．上の ans 節では ans アトムの引数と p アトムの第二引数がリンクしている．

メタ節 0 番からメタ節 1 番への矢印につけられた名前 (r1) は適用されたルールをあら

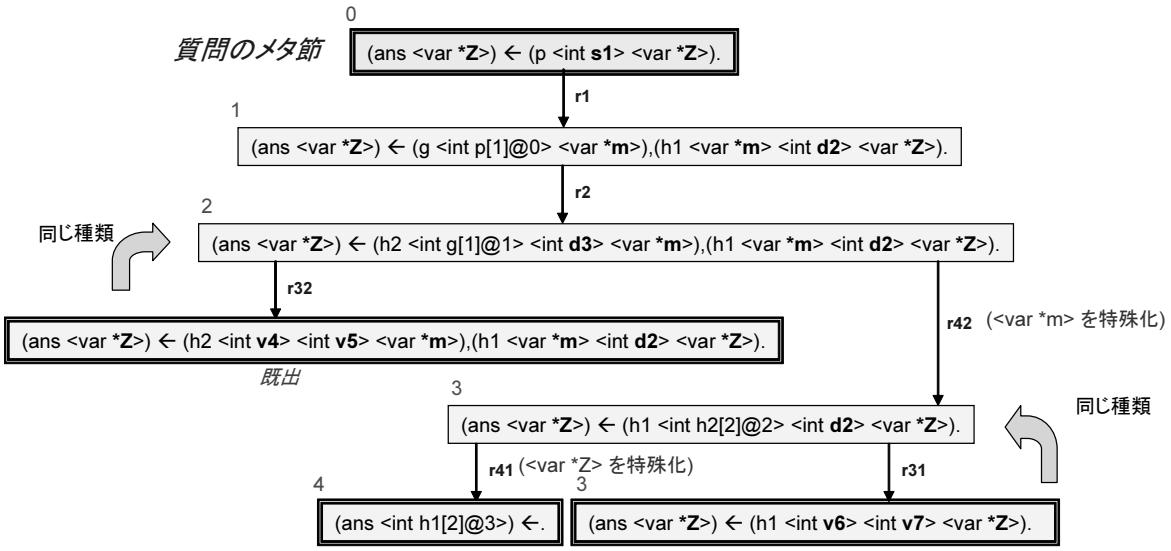


図 5.6　メタ計算

わしている．他の矢印の名前も同様である．矢印の中には「<var *m>を特殊化」のような情報を持つものがある．これは，変数 *m に対する代入が発生したことを示している．この情報は，後でルール変換の時に必要となる．

データのラベルが **a**[*n*]@*m* の形のものは，その値の由来の情報を示している．その情報とは「メタ節 *m* 番 (@*m*) のアトム **a** の第 *n* 引数の値を直接引き継いだ」という内容になっている．例えば，メタ節 4 番のラベル h1[2]@3 は，メタ節 3 番の h1 アトムの第二引数の値を引き継いでいる．この命名規則に従っていないラベル (d2,d3,v4,v5 など) は実行部で演算されるなどして新たに出現した値であり，値の由来の情報を持たない．

節の種類の分類方法は次のようにおこなっている．

二つのメタ節が同じ種類であるとは次の条件を満たすことである．

1. それらの節の間に異なる述語が存在しない．
2. 変数のリンクの仕方が同じである．
3. 変数以外のデータ型が共通している (ラベルは異なってもよい)

上記を満たさない節は異なる種類である．

例えば，メタ節 2 番とメタ節 3 番では，メタ節 2 番に含まれる h2 アトムがメタ節 3 番に含まれないため異なる種類である．また，メタ節 2 番と，それにルール r32 を適用して得られる節は，ボディアトムが全く同一の述語であり，尚且つ変数 *Z および *m のリンクの仕方も同じであり，それ以外のデータの型が同じなので同種類の節である．

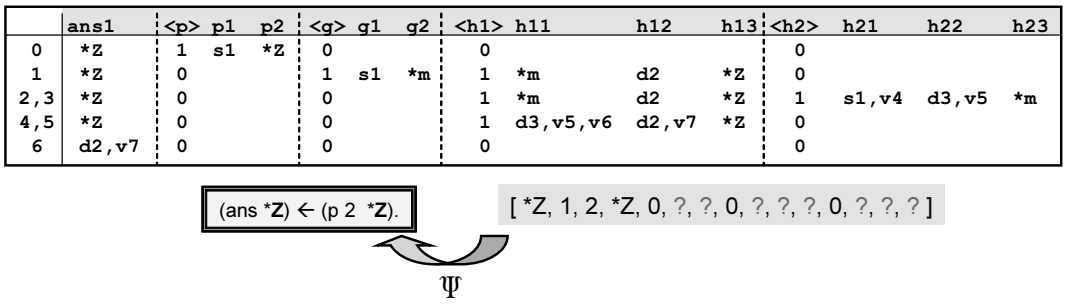


図 5.7　レジスタ

状態空間の決定

このメタ計算の結果，この例題の状態空間は次の 5 種類の節だけからなることが分かる．

```
節 4 : (ans <INT>) <-
節 3 : (ans *1) <- (h1 *2 <INT> *1)
節 2 : (ans *1) <- (h2 <INT> <INT> *2),(h1 *2 <INT> *1)
節 1 : (ans *1) <- (g <INT> *2), (h1 *2 <INT> *1)
節 0 : (ans *1) <- (p <INT> *1)
```

この有限な状態空間を表現するには 15 個のレジスタがあればよい (図 5.7)．一番上のカラムの ans1,<p>,$\cdots$, h23 がレジスタ名をあらわし，その下の 5 個のカラムはレジスタの内容をあらわしている．内容が空白になっている箇所は任意の値 (don't care) で構わないことをあらわす．図 5.7 の下段には，写像 Ψ によってレジスタの内容から確定節へ一意に変換される例が示されている．

5.3.3　レジスタの最適化

図 5.7 の空白はレジスタの内容が don't care であることをあらわしている．つまり，その状態のときはレジスタが使われていない．したがって任意の値を入れても構わない．例えばレジスタ p2 の内容を ans1 と同じにしても問題ない．同様に h13 も ans1 と同じ内容にできる．つまり ans1,p2,h13 の 3 つは一つのレジスタにまとめることができる．

このようなレジスタを見つけるには，各レジスタの内容をリストであらわし，それらの**単一化** (unification) を行ってみればよい．これを ans1,p2,h13 を例に説明する．各レジスタの内容を縦にみたとき，空白 (don't care) の場所を**無名変数**とし，それ以外の値 (変数も含む) を基礎項とみなしたリストを作成する．

```
ans1 = [ *Z, *Z, *Z, *Z, {d2,v7} ]
p2   = [ *Z, _ , _ , _ , _ ]
h13  = [ _ , *Z, *Z, *Z, _ ]
```

このリストの *Z は基礎項として扱うので変数は無名変数だけである．無名変数の場所は don't care をあらわしており，リストが単一化できるということは，レジスタの内容が同じで構わないことを意味する．

このようにして他の共有可能なレジスタを調べると，全部で 3 種類の共有レジスタがあることが分かる (下記).

```
reg1 = (p1,g1,h21)
reg2 = (ans1,p2,h13)
reg3 = (g2,h11,h23)
```

この結果，必要なレジスタの数は次の 9 個で済む.

```
reg1,reg2,reg3,reg4,reg5,<p>,<g>,<h1>,<h2>
```

5.4 レジスタ書き換えルールへの変換

メタ計算によって状態空間が明らかとなり節を表現するために必要なレジスタ数も明らかとなったので，写像 $\mathcal{F}$ と Ψ を定義して節の状態空間をレジスタの状態空間へ変換することができる．

さらにメタ計算によって変数への代入が発生する計算パスが明らかとなったので，ルールによる節の変数への代入が生じないようにルールを変更する．具体的には変数への代入が発生する部分を，その変数を含むアトムの置き換え (replace) に変換する．

この結果，図 5.4 で示された ET ルールは，図 5.8 に示すレジスタ書き換え型のルールへ変換される．これらのルールはヘッドとボディが同じ形であり，ヘッドの変数に対する代入がない．したがって一つのルールにマージすることが可能である．

5.5 ルールのマージ

ルールのマージは**スケジューリング**の働きをする．すなわち，非決定的であったルールの適用順序はマージによって決定的になる．最終的に全てのルールは一つにマージされ，そのルールの実行部が実質的な「遷移関数」となる．ここでは，前セクションで得られたレジスタ書き換えルールをマージする方法について述べる．

§5.4 の例で示したように，回路化可能なルールと確定節はプログラム変換によってレジスタ書き換えルールとレジスタ列をあらわす節に変換される．このとき，ルールと確定節は条件 (5.18) を満たすように変換されている．

```
//r1
(flgP 1),(flgG ?),(flgH1 ?),
(reg1 *x),(reg2 *y),(reg3 ?),(reg4 ?)
  ==> (flgP 0),(flgG 1),(flgH1 1),
      (reg1 *x),(reg2 *y),(reg3 *m),(reg4 0).
//r2
(flgG 1),(flgH2 ?),
(reg1 *x),(reg3 *z),(reg5 ?)
  ==> (flgG 0),(flgH2 1),
      (reg1 *x),(reg3 *z),(reg5 0).
//r31
(flgH1 1),
(reg2 *w),(reg3 *x),(reg4 *s), {(number *x),(> *x 0)}
  ==> {(:= *x1 (- *x 1)), (:= *s1 (+ *s *x))},
      (flgH1 1),
      (reg2 *w),(reg3 *x1),(reg4 *s1).
//r32
(flgH2 1),
(reg1 *x),(reg3 *w),(reg5 *s), {(number *x), (> *x 0)}
  ==> {(:= *x1 (- *x 1)), (:= *s1 (+ *s *x))},
      (flgH2 1),
      (reg1 *x1),(reg3 *w),(reg5 *s1).
//r41
(flgH1 1),
(reg2 *w),(reg3 0),(reg4 *c)
  ==> (flgH1 0),
      (reg2 *c),(reg3 0),(reg4 *c).
//r42
(flgH2 1),(flgH1 1),
(reg1 0),(reg2 *z),(reg3 *w),(reg4 *y),(reg5 *s)
  ==> (flgH2 0),(flgH1 1),
      (reg1 0),(reg2 *z),(reg3 *s),(reg4 *y),(reg5 *s).
```

図 5.8　レジスタ書き換えルール

レジスタ書き換えルールが満たしている条件

1. ルールヘッドとルールボディは同じ述語アトムを持ち，それらの引数だけが異なっている．
2. 確定節のボディアトムの中にルールヘッドの述語が必ず存在する．したがってルールは必ずマッチングする．
3. 実行部はルールのボディアトムの引数を計算するだけで，ルールヘッドの変数に対する代入を発生しない．
4. アトムの引数が全て変数であると仮定する．

(5.18)

4 番目の条件は仮定であるが，この仮定によって一般性は失われない．なぜならレジスタ書き換えルールは仮定を満たす等価なルールに容易に変換できるからである．状態空間 (確定節の空間) は固定長の節 (レジスタの列を表現した節) の空間に変換される．例えば次のような形である．

$$\mathsf{ans} \leftarrow p(t_1),\ q(t_2),\ r(t_3),\ s(t_4). \tag{5.19}$$

5.5.1 ルールのマージの例

ルールのマージとは，例えば条件 (5.18) を満たす下記の二つのルール r_1 および r_2 をマージして等価な一つのルールを生成することである．

$$r_1 : p(X_1),\{\mathrm{cond}_1\} ==> \{\mathrm{exec}_1\}, p(X_2).$$
$$r_2 : q(Y_1),\{\mathrm{cond}_2\} ==> \{\mathrm{exec}_2\}, q(Y_2).$$

これらのルールはヘッドとボディの形が同じで引数だけが異なっている．また条件 (5.18) を満たすのでルールヘッドに対する代入が発生しない．この条件の下ではルールの実行部は関数を用いて次のように表現できる．

$$\mathrm{exec}_1 \ \Leftrightarrow\ X_2 := f(X_1)$$
$$\mathrm{exec}_2 \ \Leftrightarrow\ Y_2 := g(Y_1)$$

f, g は関数である．ルール r_1, r_2 が節 (5.19) に適用されるとき，p, q の引数の値 (t_1, t_2) が置換されるだけで節の変数に対する代入は発生しない．また，節の述語アトムの数や種類が変化することもない．その理由を §3.4 で説明したルール適用のセマンティクスに沿って説明する．

1. **[節のアトムの置換]** ルールのヘッドとボディが同じ述語アトムなので，ルールは節のボディアトムを同じ述語アトムに置き換える．したがって引数以外は変化しない．
2. **[節に対する代入]** 実行部によって生じる変数代入は X_2, Y_2 に対するものだけである．これらの変数はルール適用時に**変数名変更代入**によって節に出現しないユニークな変数名に変更されている．したがって元の節の変数への代入は生じない．

もし $p(X_1)$ と $q(Y_1)$ が異なる述語アトムならば，この二つのルール r_1, r_2 をマージした結果は下記の形であらわせる．

$$p(X_1), q(Y_1), \{\mathrm{cond}_\sigma\} \texttt{ ==> } \{\mathrm{exec}_\sigma\}, p(X_2), q(Y_2).$$

このルールの作用が r_1, r_2 の二つのルールと等価になるように，うまく cond_σ および exec_σ を決めることができればよい．二つのルールと等価な 1 個のルールを生成するには $f, g, \mathrm{cond}_1, \mathrm{cond}_2, X_1, Y_1, X_2, Y_2$ を用いて新たな条件部 cond_σ と新たな実行部 exec_σ を定義すればよい．

このようなマージを繰り返し適用することで全てのルールがマージされ，等価な一つのルールが出来上がる．

5.6 マージ方法の種類と説明

条件 (5.18) を満たすルールは次の性質を持つことが容易に分かる．

- ルールヘッドの述語が必ず確定節のボディに存在しており，さらに引数が変数なのでルールヘッドは必ず確定節のボディにマッチする．したがってルールの適用可能性は条件部 cond だけで決まる．
- 実行部はルールヘッドの変数に対する代入を生じない．したがってルールは確定節の変数に対する代入を生じない．
- ルールヘッドとルールボディが同じ述語なので確定節の形 (述語の数や種類) は変わらない．確定節の引数の値だけが置き換え (replace) られる．

プログラム変換によって得られるレジスタ書き換えルールは上記の性質を持つ．この性質を持つルールは，マージする上で非常に都合がよい．以下では，この性質を持つルールに対して利用可能な 3 種類のマージ方法について説明する．

■並列的　二つのルールを同時に適用するもので，計算時間の短縮に大きく寄与する方法である．二つのルールが並列に適用可能な場合に用いることができる．

■シーケンシャル　二つのルールを連続して適用するもので，計算パスのショートカットを実現する方法である．二つのルールが連続して適用できる場合に用いることができる．

■排他的　任意の二つのルールをマージする方法である．計算時間の効率には寄与しないが，必ずマージが可能なためルールを一つにマージするのに必須の方法である．

5.6.1 並列的マージの説明

並列的マージを簡単に説明する．同時に適用可能な二つのルールを一つに合成することで適用を高速化する方法である．この方法は二つのルールが互いに独立で同時に適用しても正当性が保証される場合に利用可能である．説明には下記のルールを用いる．

$$r_{p1} : p(X),\{\mathrm{cond}_1(X)\} \Longrightarrow \{U := f(X)\}, p(U).$$
$$r_{p2} : q(Y),\{\mathrm{cond}_2(Y)\} \Longrightarrow \{V := g(Y)\}, q(V).$$

X と Y はヘッドアトムの引数である．この形の二つの ET ルール r_{p1}, r_{p2} を「同時に」適用したのと同じ効果を持つ ET ルールを生成することができる．仮定 (5.18) により，これらのルールにはヘッドの変数に対する代入がなく，実行部はボディの引数の計算だけを行う．またヘッドアトムの引数 X, Y が変数であると仮定する．この仮定によって一般性は失われない．なぜなら引数のマッチングを条件部で判定するような等価なルールに変形できるからである．この二つのルールから，等価な以下のルールを生成することができる．

$$r_p \ : \ p(X), q(Y), \{\mathrm{cond}_p\} \Longrightarrow \{\mathrm{exec}_p\}, p(U), q(V).$$

例えば cond_p と exec_p を以下のように定義したとき，ルール r_p は r_{p1} および r_{p2} の両方と等価なルールとなる．

cond_P の式　　: $(\mathrm{cond}_1(X) \vee \mathrm{cond}_2(Y))$
exec_P の手続き :
　　if $\{\ \mathrm{cond}_1(X)\ \}$ then $\{\ U{:=}f(X)\ ;\}$ else $\{\ U{:=}X\ ;\}$
　　if $\{\ \mathrm{cond}_2(Y)\ \}$ then $\{\ V{:=}g(Y)\ ;\}$ else $\{\ V{:=}Y\ ;\}$

ルール r_p が r_{p1}, r_{p2} と等価なルールであることを理解するには，$\mathrm{cond}_1(X)$ と $\mathrm{cond}_2(Y)$ の真偽の組み合わせ (全 4 通り) に対するルールの作用を考えてみればよい．例として $\mathrm{cond}_1(X)$ が成立し $\mathrm{cond}_2(Y)$ が不成立のケース (r_{p1} だけが適用可能なケース) を考える．

- このとき cond_p は真なので r_p は質問節に適用可能である．
- もし r_p を質問節に適用すると，$U{=}f(X), V{=}Y$ なので質問節のボディにある $p(X)$ は $p(f(X))$ に置き換えられ，$q(Y)$ は $q(Y)$ に置き換えられる (変化しない)．
- 結局このケースでは r_p は r_{p1} と同じ働きをする．

$\mathrm{cond}_1(X), \mathrm{cond}_2(Y)$ の他の真理値の組み合わせについても同様に等価性が示される．したがって，r_p が r_{p1}, r_{p2} の両方と等価なルールであることが分かる．

条件部と実行部の決め方

上の例で見たように，$\mathrm{cond}_1(X)$ と $\mathrm{cond}_2(Y)$ の真偽の組み合わせ (全 4 通り) に対して r_p が r_{p1}, r_{p2} と等価な働きをすればよい．そのための条件は以下のようにして求められる．

■exec$_p$ の条件:

(1) $\mathrm{cond}_1(X) \wedge \neg\mathrm{cond}_2(Y)$ のとき
このときには r_{p1} だけが適用されたのと同じ作用を持てばよい．つまり，exec_p によって代入 $\rho_{\mathrm{exec}} = \{\, U/f(X),\ V/Y \,\}$ が発生すればよい．

(2) $\neg\mathrm{cond}_1(X) \wedge \mathrm{cond}_2(Y)$ のとき
このときには r_{p2} だけが適用されたのと同じ作用を持てばよい．つまり，exec_p によって代入 $\rho_{\mathrm{exec}} = \{\, U/X,\ V/g(Y) \,\}$ が発生すればよい．

(3) $\mathrm{cond}_1(X) \wedge \mathrm{cond}_2(Y)$ のとき
このときには r_{p1} と r_{p2} の両方が適用可能である．更に仮定により r_{p1} と r_{p2} は互いに干渉せず同時に適用可能である (**並列的なマージ**が可能)．つまり，exec_p によって代入 $\rho_{\mathrm{exec}} = \{U/f(X),\ V/g(Y)\,\}$ が発生すればよい．

(4) $\neg\mathrm{cond}_1(X) \wedge \neg\mathrm{cond}_2(Y)$ のとき
このときには r_{p1} も r_{p2} も適用されない．したがって r_p も適用されないよう cond_p が偽になればよい．ルールが適用されないので exec_p の内容は任意で構わない．

■cond$_p$ の条件: 上記の 4 通りの組み合わせの中でルール r_p が適用可能なケースは (1), (2), (3) の 3 通りである．したがって cond_p は，これら 3 つの前提条件の論理和となる．

$$\mathrm{cond}_p \stackrel{\mathrm{def}}{=} (\mathrm{cond}_1(X) \wedge \neg\mathrm{cond}_2(Y)) \vee (\neg\mathrm{cond}_1(X) \wedge \mathrm{cond}_2(Y)) \vee (\mathrm{cond}_1(X) \wedge \mathrm{cond}_2(Y))$$

これは $(\mathrm{cond}_1(X) \vee \mathrm{cond}_2(Y))$ に等しい (証明は容易なので省略する)．つまり r_p が適用可能なのはルール r_{p1} と r_{p2} の少なくとも一つが適用可能のときである．

5.6.2 シーケンシャルなマージの説明

二つのルールが連続して適用可能な場合に効果的な合成方法である．この方法は並列的なマージができないルール同士にも利用できる．説明には下記のルールを用いる．

$$\begin{aligned} r_{s1} : p(X), r(Z_1), \{\mathrm{cond}_1(X, Z_1)\} \\ \qquad\qquad ==>\ \ \{U := f_p(X, Z_1);\ W_1 := f_r(X, Z_1)\},\ p(U), r(W_1). \\ r_{s2} : q(Y), r(Z_2), \{\mathrm{cond}_2(Y, Z_2)\} \\ \qquad\qquad ==>\ \ \{V := g_q(Y, Z_2);\ W_2 := g_r(Y, Z_2)\},\ q(V), r(W_2). \end{aligned}$$

シーケンシャルなマージとは，確定節に対して r_{s1} と r_{s2} を連続して適用したのと同じ作用を持つルール r_s を生成する方法である．そのルールは下記の形をしている．

$$r_s\ :\ p(X), q(Y), r(Z),\ \{\mathrm{cond}_s\}\ \ ==>\ \ \{\mathrm{exec}_s\},\ p(U), q(V), r(W). \qquad (5.20)$$

ヘッド (およびボディ) は二つのルールの述語の和集合となっており，cond_s および exec_s を適切に決定することで r_{s1}, r_{s2} と等価なルールにできる．この考え方を以下に示す．

シーケンシャルマージの考え方

ET ルールの性質から，適用可能なルールが複数ある場合はどれを適用しても正当性は保証される (適用の非決定性)．そこで次のような戦略でルール選択を行ったのと等価な作用を持つようにルール r_s を生成する．

(1) 確定節 s_n に r_{s1} と r_{s2} の両方が適用可能ならば，通常は r_{s1} を優先して選択する．
  (1-a) ただし，s_n に r_{s1} を適用した直後の確定節 s_{n+1} に r_{s2} が適用可能な場合には **必ず** s_{n+1} に r_{s2} を適用する (r_s は $r_{s1} \rightarrow r_{s2}$ の連続適用と等価)．
  (1-b) s_{n+1} に r_{s2} が適用できない場合は通常のルール選択に戻る (r_s は r_{s1} と等価)．
(2) もし s_n に r_{s1} が適用できない場合，r_{s2} が適用できるならば s_n に r_{s2} を適用してもよい (r_s は r_{s2} と等価)．
(3) r_{s1} も r_{s2} も適用できない場合は他のルールが選択される (r_s は適用条件が偽になっていればよい)．

条件部と実行部の決め方

上記の (1-a), (1-b), (2), (3) の 4 通りについて cond_s と exec_s が満たすべき条件を考えればよい．

■**exec_s の条件**: 書き換え対象の節を次の形であらわし，これをもとに exec_s が満たすべき条件を議論する．

$$\mathsf{ans} \leftarrow p(t_x), q(t_y), r(t_z), \Delta(\Lambda).$$

ここで $\Delta(\Lambda)$ は任意のアトム集合をあらわしている．ただし p, q, r アトムを含んでいないものとする．

(1-a) r_{s1} が適用可能で，その適用結果に r_{s2} が適用可能な場合の条件を考える．
1. まず r_{s1} が適用可能なのでマッチング代入 θ_{1a} =$\{X/t_x,\ Z_1/t_z\}$ が存在し，r_{s1} の条件部 $\mathrm{cond}_1(X, Z_1)\theta_{1a} = \mathrm{cond}_1(t_x, t_z)$ が真になる．
2. r_{s1} を適用すると，節のアトム $\{p(t_x),\ r(t_z)\}$ は $\{p(f_p(t_x, t_z)),\ r(f_r(t_x, t_z))\}$ に置き換わる．
3. 上記の結果に r_{s2} が適用可能なのでマッチング代入 $\theta'_{1a} = \{Y/t_y,\ Z_2/f_r(t_x, t_z)\}$ に対して $\mathrm{cond}_2(Y, Z_2)\theta'_{1a} = \mathrm{cond}_2(t_y, f_r(t_x, t_z))$ が真である．
4. r_{s2} を適用すると，節のアトム $\{q(t_y),\ r(f_r(t_x, t_z))\}$ は $\{q(g_q(t_y, f_r(t_x, t_z))),\ r(g_r(t_y, f_r(t_x, t_z)))\}$ に置き換わる．
5. 以上から，r_s は $\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, f_r(t_x, t_z))$ のとき $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(f_p(t_x, t_z)), q(g_q(t_y, f_r(t_x, t_z))), r(g_r(t_y, f_r(t_x, t_z)))\}$ に置き換える．

$$\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, f_r(X, Z))$$
$$\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ U/f_p(X, Z),\ V/g_q(Y, f_r(X, Z)),\ W/g_r(Y, f_r(X, Z)) \right\}$$

(1-b) r_{s1} が適用可能で，その適用結果に r_{s2} が適用できない場合の条件を考える．

1. r_{s1} が適用可能なので $\mathrm{cond}_1(t_x, t_z)$ が真である．
2. r_{s1} を適用すると節の $p(t_x), r(t_z)$ が $p(f_p(t_x, t_z)), r(f_r(t_x, t_z))$ に置き換わる．
3. この適用結果に r_{s2} が適用できないので $\mathrm{cond}_2(t_y, f_r(t_x, t_z))$ は偽である．
4. 以上から，r_s は $\mathrm{cond}_1(t_x, t_z) \wedge \neg\mathrm{cond}_2(t_y, f_r(t_x, t_z))$ のとき $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(f_p(t_x, t_z)), q(t_y), r(f_r(t_x, t_z))\}$ に置き換える．

$$\mathrm{cond}_1(X, Z) \wedge \neg\mathrm{cond}_2(Y, f_r(X, Z)) \\ \Rightarrow \ \rho_{\mathrm{exec}} = \left\{ U/f_p(X, Z),\ V/Y,\ W/f_r(X, Z) \right\}$$

(2) r_{s1} が適用できず，r_{s2} が適用可能な場合の条件を考える．

1. r_{s1} が適用できないので $\mathrm{cond}_1(t_x, t_z)$ が偽である．
2. r_{s2} が適用可能なので $\mathrm{cond}_2(t_y, t_z)$ が真である．
3. r_{s2} を適用すると節の $q(t_y), r(t_z)$ が $q(g_q(t_y, t_z)), r(g_r(t_y, t_z))$ に置き換わる．
4. 以上から，r_s は $\neg\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, t_z)$ のとき $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(t_x), q(g_q(t_y, t_z)), r(g_r(t_y, t_z))\}$ に置き換える．

$$\neg\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, Z) \\ \Rightarrow \ \rho_{\mathrm{exec}} = \left\{ U/X,\ V/g_q(Y, Z),\ W/g_r(Y, Z) \right\}$$

(3) r_{s1} と r_{s2} が適用できない場合 $(\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, t_z))$，$r_s$ も適用されない．

■cond$_s$ の条件: r_s が適用されるのは上記 (1-a), (1-b), (2) のときである．したがって，これらの前提条件の論理和となる．

$$\begin{aligned} \mathrm{cond}_s \stackrel{\mathrm{def}}{=} & (\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, f_r(X, Z))) \\ & \vee (\mathrm{cond}_1(X, Z) \wedge \neg\mathrm{cond}_2(Y, f_r(X, Z))) \\ & \vee (\neg\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, Z)) \end{aligned}$$

これは $(\mathrm{cond}_1(X, Z) \vee \mathrm{cond}_2(Y, Z))$ に等しい．証明は容易なので省略する．

■実装例: 下記は cond$_s$ および exec$_s$ の条件を満たす実装例である．

```
condₛ の式      : (cond₁(X,Z) ∨ cond₂(Y,Z))
execₛ の手続き : if cond₁(X,Z) then
                   { U := f_p(X,Z); T := f_r(X,Z); }
                 else
                   { U := X; T := Z; };
                 if cond₂(Y,T) then
                   { V := g_q(Y,T); W := g_r(Y,T); }
                 else
                   { V := Y; W := T; };
```

5.6.3 排他的マージの説明

これは二つのルールを一つにすることが目的の合成方法である．合成されたルールによって計算効率はほとんど向上しないが，任意の 2 ルールを必ずマージできるので回路生成には重要な方法である．説明には下記のルールを用いる．

$$
\begin{aligned}
r_{e1} : p(X), r(Z_1), \{\mathrm{cond}_1(X, Z_1)\} \\
\qquad\qquad ==> \ \left\{U := f_p(X, Z_1);\ W_1 := f_r(X, Z_1)\right\},\ p(U), r(W_1). \\
r_{e2} : q(Y), r(Z_2), \{\mathrm{cond}_2(Y, Z_2)\} \\
\qquad\qquad ==> \ \left\{V := g_q(Y, Z_2);\ W_2 := g_r(Y, Z_2)\right\},\ q(V), r(W_2).
\end{aligned}
$$

排他的なマージ方法は，確定節に対して r_{e1} と r_{e2} のどちらかひとつを適用したのと同じ作用を持つルール r_e を生成する方法である．生成されたルールは下記の形をしている．

$$
r_e \ :\ p(X), q(Y), r(Z),\ \{\mathrm{cond}_s\}\ \ ==>\ \ \{\mathrm{exec}_s\},\ p(U), q(V), r(W). \tag{5.21}
$$

ヘッド (およびボディ) は二つのルールの述語の和集合となっており，cond_e および exec_e を適切に決定することで r_{e1}, r_{e2} と等価なルールにできる．この考え方を以下に示す．

排他的マージの考え方

排他的なマージの考え方はシンプルである．マージされたルール r_e は，$\mathrm{cond}_1, \mathrm{cond}_2$ の真偽に応じて次の性質を満たせばよい．

(1) もし r_{e1} が適用可能 (cond_1) で r_{e2} が適用できない ($\neg\mathrm{cond}_2$) 場合，マージされたルールは r_{e1} と等価な書き換えをおこなえばよい．
(2) もし r_{e1} が適用できず ($\neg\mathrm{cond}_1$)， r_{e2} が適用可能 (cond_2) ならば，マージされたルールは r_{e2} と等価な書き換えをおこなえばよい．
(3) もし r_{e1} と r_{e2} の両方が適用可能 ($\mathrm{cond}_1 \wedge \mathrm{cond}_2$) ならば，ET ルールの適用の非決定性の性質により，マージされたルールは r_{e1} または r_{e2} のどちらかと等価な書き換えをおこなえばよい．どちらのルールを選択しても構わない．適用可能なルールはどれを適用しても正当性に影響を与えないからである．
(4) r_{e1}, r_{e2} ともに適用できない場合は r_e も適用できなければよい．

■exec_e の条件: exec_e の満たすべき条件は，上記の考え方で示した 4 つのケースに分けて考えればよい．書き換え対象の節を次の形であらわし，これをもとに exec_e が満たすべき条件を議論する．

$$
\mathsf{ans} \ \leftarrow\ p(t_x), q(t_y), r(t_z), \Delta(\Lambda).
$$

ここで $\Delta(\Lambda)$ は任意のアトム集合をあらわしている．ただし p, q, r アトムを含んでいないものとする．

(1) $\mathrm{cond}_1(t_x, t_z) \wedge \neg\mathrm{cond}_2(t_y, t_z)$ のとき
マージルール r_e が r_{e1} と同じ作用をすればよい．すなわち r_e は $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(f_p(t_x, t_z)), q(t_y), r(f_r(t_x, t_z))\}$ に置き換える．

$$\mathrm{cond}_1(X, Z) \wedge \neg\mathrm{cond}_2(Y, Z) \Rightarrow \rho_{\mathrm{exec}} = \{\, U/f_1(X, Z),\ V/Y,\ W/f_2(X, Z) \,\}$$

(2) $\neg\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, t_z)$ のとき
マージルール r_e が r_{e2} と同じ作用をすればよい．すなわち r_e は $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(t_x), q(g_q(t_y, t_z)), r(t_r(t_y, t_z))\}$ に置き換える．

$$\mathrm{cond}_1(X, Z) \wedge \neg\mathrm{cond}_2(Y, Z) \Rightarrow \rho_{\mathrm{exec}} = \{\, U/X,\ V/g_1(Y, Z),\ W/g_2(Y, Z) \,\}$$

(3) $\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, t_z)$ のとき
r_e は r_{e1} と r_{e2} のどちらの作用でもよい．すなわち r_e は $\{p(t_x), q(t_y), r(t_z)\}$ を $\{p(f_p(t_x, t_z)), q(t_y), r(f_r(t_x, t_z))\}$ または $\{p(t_x), q(g_q(t_y, t_z)), r(t_r(t_y, t_z))\}$ に置き換える．

$$\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, Z) \Rightarrow \rho_{\mathrm{exec}} = \{\, U/f_1(X, Z),\ V/Y,\ W/f_2(X, Z) \,\}$$
または
$$\rho_{\mathrm{exec}} = \{\, U/X,\ V/g_1(Y, Z),\ W/g_2(Y, Z) \,\}$$

(4) $\neg\mathrm{cond}_1(t_x, t_z) \wedge \mathrm{cond}_2(t_y, t_z)$ のとき
このとき r_e が適用されないように条件部を決める．適用されないので実行部は任意でよい．

条件 (3) の最初の代入を採用すれば ルール r_{e1} 優先の合成ルールになり，二番目を採用すれば ルール r_{e2} 優先の合成ルールになる．また，t_x, t_y, t_z のコンテクストに応じて採用する代入を変えても構わない．コンテクストに応じて r_{e1}, r_{e2} の選択の仕方を変えるのと同じ理由による．

■$\mathbf{cond}_e$ の条件: マージされたルール r_e が適用可能となるべきケースは上記の (1), (2), (3) である．したがって cond_e はこれらの前提条件の論理和として定義される．

$$\mathrm{cond}_e \stackrel{\mathrm{def}}{=} (\mathrm{cond}_1(X, Z) \wedge \neg\mathrm{cond}_2(Y, Z)) \vee (\neg\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, Z)) \vee (\mathrm{cond}_1(X, Z) \wedge \mathrm{cond}_2(Y, Z))$$

この式は $(\mathrm{cond}_1(X, Z) \vee \mathrm{cond}_2(Y, Z))$ に等しいことが容易に証明できる．

■実装例:

$\mathrm{cond}_e : (\mathrm{cond}_1(X,Z) \vee \mathrm{cond}_2(Y,Z))$
exec_e :
if $\mathrm{cond}_1(X,Z)$ then $\{ U := f_1(X,Z)\ ;\ V := Y\ ;\ W := f_2(X,Z); \}$
else $\{ U := X\ ;\ V := g_1(Y,Z)\ ;\ W := g_2(Y,Z); \};$

5.7 各マージ方法の一般的な表現

ここでは「並列的」「シーケンシャル」「排他的」の各マージ手法について，ルールのヘッドアトム集合が任意の場合についての議論を行う．まず最初に簡略化された記法を導入する．その後，その記法に基づいて各マージ手法を議論する．

5.7.1 簡略記法の導入

一般的な議論をするにあたり，見やすさのため簡略化された記法を導入する．一般に条件 (5.18) を満たすルールは次の形式をしている．

$$p_1(X_1),\ldots,\ p_n(X_n),\ \{\mathrm{cond}\,(X_1,\ldots,\ X_n)\}$$
$$==>\ \{\mathrm{exec}\,(X_1,\ldots,\ X_n,\ U_1,\ldots,\ U_n)\},\ p_1(U_1),\ldots,\ p_n(U_n). \qquad (5.22)$$

(where $n \geq 1$)

各 p_i は述語名，X_i は各ヘッドアトムの引数，U_i は各ボディアトムの引数である．**以降の説明ではアトムの引数 X_i, U_i が全て変数であると仮定する**．この仮定によって一般性は失われない．なぜなら，そのような等価なルールへ容易に変換できるからである．

このルールはヘッドとボディが同じ述語 $p_1, p_2, \ldots, p_n$ を持っている．また，条件 (5.18) よりルールの実行部 exec がヘッドの変数に対する代入を行わない．実行部はボディの引数 U_i の値を計算するだけであり，その値はヘッドの引数 $(X_1, \ldots, X_n)$ だけで決まる．すなわちボディの引数 U_i の計算は関数 $f_i(X_1, \ldots, X_n)$ としてあらわすことができ，実行部はその関数の値を変数 U_i へ代入する手続きに等しい．

$$\mathrm{exec}\ \equiv\ \{ U_1 := f_1(X_1, \ldots, X_n);\ \cdots\ ;\ U_n := f_n(X_1, \ldots, X_n)\ \} \qquad (5.23)$$

関数 $f_i(X_1, \ldots, X_n)$ の返値には変数も含まれる[*3]．

形式 (5.22) の ET ルールを**アトム集合**と**ベクトル** (あるいは配列) による簡略化された表記であらわす．アトム集合を白抜き文字 $\mathbb{A}$ であらわし，ベクトルをアルファベットの

[*3] ヘッドの変数がそのままボディの引数へ渡されることもあるため，関数 $f_i(X_1, \ldots, X_n)$ のドメインには変数が含まれる．

ボールド体 (例えば $\mathbf{X}$ など) であらわすことにする．このとき，ET ルール (5.22) は次の簡略記法 (5.24) であらわすことができる．

$$\mathbb{A}_{\mathbf{P}}(\mathbf{X}),\ \{\,\mathrm{cond}(\mathbf{X})\,\}\ \texttt{==>}\ \{\,\mathbf{U} := \mathbf{F}(\mathbf{X});\,\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}). \tag{5.24}$$

ここで $\mathbf{X}, \mathbf{U}$ はそれぞれヘッドアトムの引数からなるベクトル，およびボディアトムの引数からなるベクトルをあらわす．

$$\mathbf{X} = \langle X_1, \ldots, X_n \rangle$$
$$\mathbf{U} = \langle U_1, \ldots, U_n \rangle$$

ベクトルの i 番目の要素を指定する必要がある場合は，C 言語などの配列のインデックス指定と同様にベクトルの後ろに $[\,i\,]$ を付けてあらわす．

$$\text{例：}\ \mathbf{X}[\,i\,]\ =\ X_i$$

$\mathbb{A}_{\mathbf{P}}$ のサフィックス $\mathbf{P}$ は述語のベクトルをあらわす．

$$\mathbf{P} = \langle p_1, \ldots, p_n \rangle$$

$\mathbb{A}_{\mathbf{P}}(\mathbf{X})$ はルールのヘッドアトム集合をあらわしており，以下のように定義される．

$$\begin{aligned}\mathbb{A}_{\mathbf{P}}(\mathbf{X}) &\stackrel{\text{def}}{=} \{\, p(x) \mid p = \mathbf{P}[\,i\,],\ x = \mathbf{X}[\,i\,],\ 1 \le i \le n \,\} \\ &= \{\, p_1(X_1), p_2(X_2), \ldots, p_n(X_n) \,\}\end{aligned}$$

同様に $\mathbb{A}_{\mathbf{P}}(\mathbf{U})$ はボディアトム集合である．なお，アトムの引数の値を無視して考えるときには単に $\mathbb{A}_{\mathbf{P}}$ のように記述する．これは述語のベクトル $\mathbf{P}$ の要素の集合と同一視できる．

$$\mathbb{A}_{\mathbf{P}} = \{\, p \mid p = \mathbf{P}[\,i\,],\ 1 \le i \le n \,\}$$

ルールのヘッドとボディが同じ述語の集合 $\mathbb{A}_{\mathbf{P}}$ になっているため，この形式のルールは確定節の形を変えない．確定節のアトムの引数だけを書き換える．

簡略記法 (5.24) の $\mathbf{F}(\mathbf{X})$ は，関数 $f_i(\mathbf{X})$ $(i{=}1, 2, \ldots, n)$ からなるベクトルをあらわす．形式的に全ての関数が同じ引数 $\mathbf{X}$ を持つように表記する[*4]．

$$\mathbf{F}(\mathbf{X}) = \langle f_1(\mathbf{X}), \ldots, f_n(\mathbf{X}) \rangle$$

そしてベクトル代入手続き $(\,\mathbf{U} := \mathbf{F}\,(\mathbf{X})\,)$ は各要素ごとの代入手続きをあらわしており，実行部 (5.23) の簡潔な記述になっている．

$$\{\,\mathbf{U} := \mathbf{F}\,(\mathbf{X})\,;\,\}\ \Leftrightarrow\ \{U_1 := f_1(\mathbf{X});\ \ldots\ U_n := f_n(\mathbf{X});\}$$

[*4] 個々の f_i が $\mathbf{X}$ の全ての要素を使うとは限らない．しかし，引数の表記を統一して見易さを優先する．

5.7.2 並列的マージ方法の一般的表現

並列的なマージ方法を一般的な形で表現する．いま，条件 (5.18) を満たす二つのルール r_{11}, r_{12} が与えられたとする．

$$r_{11} \ : \ \mathbb{A}_{\mathbf{P}}(\mathbf{X}),\ \{\mathrm{cond}_1(\mathbf{X})\} \ \Longrightarrow \ \{\mathbf{U} := \mathbf{F}(\mathbf{X})\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}).$$
$$r_{12} \ : \ \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}),\ \{\mathrm{cond}_2(\mathbf{Y})\} \ \Longrightarrow \ \{\mathbf{V} := \mathbf{G}(\mathbf{Y})\},\ \mathbb{A}_{\mathbf{Q}}(\mathbf{V}).$$

ルールのヘッドアトム集合には空集合が許されないため $\mathbb{A}_{\mathbf{P}}(\mathbf{X})$ および $\mathbb{A}_{\mathbf{Q}}(\mathbf{Y})$ は空集合ではないとする．書き換え対象の確定節の表現には §3.4 (ET ルールのセマンティクス) で説明した式 (3.10) を用いる．

$$H \ \leftarrow \ \mathbb{B}.$$

ルール r_{11} が確定節に適用可能である条件は次を満たす代入 θ_1 が存在することである．

$$\exists\theta_1 \left\{ (\mathbb{A}_{\mathbf{P}}(\mathbf{X})\theta_1 \subseteq \mathbb{B}) \ \wedge \ (\mathrm{cond}(\mathbf{X})\theta_1 = \mathrm{true}) \right\}$$

このルールを適用した結果の確定節は以下のようになる．

$$H \ \leftarrow \ \mathbb{A}_{\mathbf{P}}(\mathbf{U})\theta_1\rho_1 \ \cup \ \mathbb{B}'$$
$$\text{ただし} \quad \rho_1 = \{\, \mathbf{U}/\mathbf{F}(\mathbf{X}) \,\}, \quad \mathbb{B}' = \mathbb{B} \setminus \mathbb{A}_{\mathbf{P}}(\mathbf{U})\theta_1\rho_1$$

同様に r_{12} の適用条件と書き換え結果は次のようになる．

$$\exists\theta_2 \left\{ (\mathbb{A}_{\mathbf{Q}}(\mathbf{Y})\theta_2 \subseteq \mathbb{B}) \ \wedge \ (\mathrm{cond}(\mathbf{Y})\theta_2 = \mathrm{true}) \right\}$$
$$H \ \leftarrow \ \mathbb{A}_{\mathbf{Q}}(\mathbf{V})\theta_2\rho_2 \ \cup \ \mathbb{B}''$$
$$\text{ただし} \quad \rho_2 = \{\, \mathbf{V}/\mathbf{G}(\mathbf{Y}) \,\}, \quad \mathbb{B}'' = \mathbb{B} \setminus \mathbb{A}_{\mathbf{Q}}(\mathbf{V})\theta_2\rho_2$$

以上の前提知識から，並列的なマージが可能な条件を考える．まず仮定から $\mathbf{X}, \mathbf{Y}$ は変数であり，条件 (5.18) の 2. により確定節に必ずマッチングする．したがってルールが適用できるかどうかは，そのルールの条件部だけで判断できる．

ルール r_{11} (または r_{12}) はアトムの「置き換え」ルールになっている．単にアトム $\mathbb{A}_{\mathbf{P}}(\mathbf{X})$ (または $\mathbb{A}_{\mathbf{Q}}(\mathbf{Y})$) をアトム $\mathbb{A}_{\mathbf{P}}(\mathbf{U})$ (または $\mathbb{A}_{\mathbf{Q}}(\mathbf{V})$) に置き換えているにすぎない．節の変数に対する代入は発生しない．このとき r_{11} と r_{12} が同時に適用できるためには，条件部 $\mathrm{cond}_1, \mathrm{cond}_2$ が両方とも真のときに互いに同じアトムを置き換えなければよい．つまり同じ述語アトムを持たなければよい．

並列的なマージが可能な条件

$$\mathbb{A}_{\mathbf{P}} \cap \mathbb{A}_{\mathbf{Q}} = \emptyset$$

このとき，この二つのルールから等価な以下のルールを生成することができる．

$$r_P \ : \ \mathbb{A}_{\mathbf{P}}(\mathbf{X}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}),\ \{\mathrm{cond}_P\} \ \Longrightarrow \ \{\mathrm{exec}_P\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{V}). \tag{5.25}$$

cond_P と exec_P の決め方は「並列的マージの説明」(§5.6.1) と同様の考え方に基づいて議論できる．条件部 $\mathrm{cond}_1, \mathrm{cond}_2$ の真偽の組み合わせ (全 4 通り) に対して合成ルール r_p が r_{11}, r_{12} と等価な作用をすればよい．

■**exec**$_P$： §5.6.1 と同じ議論により，実行部 exec_P は下記の 3 条件を満たすような代入 ρ_{exec} を発生させる手続きであればよい．

1. $\mathrm{cond}_1(\mathbf{X}) \wedge \neg\mathrm{cond}_2(\mathbf{Y}) \Rightarrow \rho_{\mathrm{exec}}=\{\, \mathbf{U}/\mathbf{F}(\mathbf{X}),\ \mathbf{V}/\mathbf{Y} \,\}$
2. $\neg\mathrm{cond}_1(\mathbf{X}) \wedge \mathrm{cond}_2(\mathbf{Y}) \Rightarrow \rho_{\mathrm{exec}}=\{\, \mathbf{U}/\mathbf{X},\ \mathbf{V}/\mathbf{G}(\mathbf{Y}) \,\}$
3. $\mathrm{cond}_1(\mathbf{X}) \wedge \mathrm{cond}_2(\mathbf{Y}) \Rightarrow \rho_{\mathrm{exec}}=\{\, \mathbf{U}/\mathbf{F}(\mathbf{X}),\ \mathbf{V}/\mathbf{G}(\mathbf{Y}) \,\}$
4. 上記以外 ⇒ 実行部は任意

■**cond**$_P$： 上記のうちルール r_P が適用されるべきケースは 1. 2. 3. の 3 通りである．cond_P はこれらの前提条件の論理和として定義すればよい．

$$\mathrm{cond}_P \stackrel{\mathrm{def}}{=} (\mathrm{cond}_1(\mathbf{X}) \wedge \neg\mathrm{cond}_2(\mathbf{Y})) \vee (\neg\mathrm{cond}_1(\mathbf{X}) \wedge \mathrm{cond}_2(\mathbf{Y})) \vee (\mathrm{cond}_1(\mathbf{X}) \wedge \mathrm{cond}_2(\mathbf{Y}))$$

これは $(\mathrm{cond}_1(\mathbf{X}) \vee \mathrm{cond}_2(\mathbf{Y}))$ に等しい．

■**実装例**: 上記の条件を満たすには，例えば下記のように実装すればよい．

$\mathrm{cond}_P : (\mathrm{cond}_1(\mathbf{X}) \vee \mathrm{cond}_2(\mathbf{Y}))$
exec_P :
　　if $\{\mathrm{cond}_1(\mathbf{X})\}$ then $\{\, \mathbf{U} := \mathbf{F}(\mathbf{X})\ ; \}$ else $\{\, \mathbf{U} := \mathbf{X}\ ; \}$
　　if $\{\mathrm{cond}_2(\mathbf{Y})\}$ then $\{\, \mathbf{V} := \mathbf{G}(\mathbf{Y})\ ; \}$ else $\{\, \mathbf{V} := \mathbf{Y}\ ; \}$

5.7.3 シーケンシャルなマージ方法の一般的表現

シーケンシャルなマージ方法を一般的な形で表現する．いま，条件 (5.18) を満たす二つのルール r_{21}, r_{22} が与えられたとする．ただし述語集合 $\mathbb{A}_{\mathbf{P}}, \mathbb{A}_{\mathbf{Q}}, \mathbb{A}_{\mathbf{R}}$ は互いに素である．

$$
\begin{aligned}
r_{21} : \mathbb{A}_{\mathbf{P}}(\mathbf{X}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}_1),\ \{\mathrm{cond}_1(\mathbf{X},\mathbf{Z}_1)\} \\
\quad ==> \left\{ \mathbf{U} := \mathbf{F}_p(\mathbf{X},\mathbf{Z}_1);\ \mathbf{W}_1 := \mathbf{F}_r(\mathbf{X},\mathbf{Z}_1);\ \right\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}_1).
\end{aligned}
$$

$$
\begin{aligned}
r_{22} : \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}_2),\ \{\mathrm{cond}_2(\mathbf{Y},\mathbf{Z}_2)\} \\
\quad ==> \left\{ \mathbf{V} := \mathbf{G}_q(\mathbf{Y},\mathbf{Z}_2);\ \mathbf{W}_2 := \mathbf{G}_r(\mathbf{Y},\mathbf{Z}_2);\ \right\},\ \mathbb{A}_{\mathbf{Q}}(\mathbf{V}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}_2).
\end{aligned}
$$

二つのルールは共通の述語 $\mathbb{A}_{\mathbf{R}}$ と共通でない述語 $\mathbb{A}_{\mathbf{P}}, \mathbb{A}_{\mathbf{Q}}$ を持つ．ルールヘッドには空集合が許されないので $(\mathbb{A}_{\mathbf{P}} \cup \mathbb{A}_{\mathbf{R}})$ と $(\mathbb{A}_{\mathbf{Q}} \cup \mathbb{A}_{\mathbf{R}})$ は空集合でない．シーケンシャルなマージとは，r_{21}, r_{22} の順に連続適用するのと同じ作用を持つルールを生成する方法である．そのマージされたルールは以下の形になる．

$$
r_S : \mathbb{A}_{\mathbf{P}}(\mathbf{X}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}),\ \{\mathrm{cond}_S\} ==> \{\mathrm{exec}_S\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{V}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}). \tag{5.26}
$$

cond_S と exec_S の決め方は「シーケンシャルなマージの説明」(§5.6.2) と同様の考え方に基づいて議論できる．すなわち条件部 $\mathrm{cond}_1, \mathrm{cond}_2$ の真偽の組み合わせ (この場合は複雑な組み合わせになる) に対して合成ルール r_s が r_{21}, r_{22} と等価な作用をすればよい．

■**exec**$_S$ ： §5.6.2 と同じ議論により，実行部 exec_S は下記の 3 条件を満たすような代入 ρ_{exec} を発生させる手続きであればよい．

1. $\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z}))$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{F}_p(\mathbf{X},\mathbf{Z}),\ \mathbf{V}/\mathbf{G}_q(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z})),\ \mathbf{W}/\mathbf{G}_r(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z})) \right\}$
2. $\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \neg\mathrm{cond}_2(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z}))$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{F}_p(\mathbf{X},\mathbf{Z}),\ \mathbf{V}/\mathbf{Y},\ \mathbf{W}/\mathbf{F}_r(\mathbf{X},\mathbf{Z}) \right\}$
3. $\neg\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z})$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{X},\ \mathbf{V}/\mathbf{G}_q(\mathbf{Y},\mathbf{Z}),\ \mathbf{W}/\mathbf{G}_r(\mathbf{Y},\mathbf{Z}) \right\}$
4. 上記以外 $\Rightarrow$ 実行部は任意

■**cond**$_S$ ： 上記のうちルール r_S が適用されるべきケースは 1. 2. 3. の 3 通りである．cond_S はこれらの前提条件の論理和として定義すればよい．

$$
\begin{aligned}
\mathrm{cond}_S \stackrel{\mathrm{def}}{=} & (\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z}))) \vee (\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \neg\mathrm{cond}_2(\mathbf{Y},\mathbf{F}_r(\mathbf{X},\mathbf{Z}))) \\
& \vee (\neg\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))
\end{aligned}
$$

これは $(\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))$ に等しい．

■実装例： 上記の条件を満たすには，例えば下記のように実装すればよい．

cond_S : $(\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))$
exec_S :
  if $\{\mathrm{cond}_1(\mathbf{X},\mathbf{Z})\}$ then
    $\{\mathbf{U} := \mathbf{F}_p(\mathbf{X},\mathbf{Z});\ \mathbf{T} := \mathbf{F}_r(\mathbf{X},\mathbf{Z});\}$
    else
    $\{\mathbf{U} := \mathbf{X};\ \mathbf{T} := \mathbf{Z};\};$

  if $\{\mathrm{cond}_2(\mathbf{Y},\mathbf{T})\}$ then
    $\{\mathbf{V} := \mathbf{G}_q(\mathbf{Y},\mathbf{T});\ \mathbf{W} := \mathbf{G}_r(\mathbf{Y},\mathbf{T});\}$
    else
    $\{\mathbf{V} := \mathbf{Y};\ \mathbf{W} := \mathbf{T};\};$

5.7.4 排他的マージ方法の一般的表現

排他的マージ方法を一般的な形で表現する．いま，条件 (5.18) を満たす二つのルール r_{31}, r_{32} が与えられたとする．ただし述語集合 $\mathbb{A}_{\mathbf{P}}, \mathbb{A}_{\mathbf{Q}}, \mathbb{A}_{\mathbf{R}}$ は互いに素である．

$$r_{31} : \mathbb{A}_{\mathbf{P}}(\mathbf{X}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}_1),\ \{\mathrm{cond}_1(\mathbf{X},\mathbf{Z}_1)\} \\ ==> \left\{ \mathbf{U} := \mathbf{F}_p(\mathbf{X},\mathbf{Z}_1);\ \mathbf{W}_1 := \mathbf{F}_r(\mathbf{X},\mathbf{Z}_1);\ \right\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}_1).$$

$$r_{32} : \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}_2),\ \{\mathrm{cond}_2(\mathbf{Y},\mathbf{Z}_2)\} \\ ==> \left\{ \mathbf{V} := \mathbf{G}_q(\mathbf{Y},\mathbf{Z}_2);\ \mathbf{W}_2 := \mathbf{G}_r(\mathbf{Y},\mathbf{Z}_2);\ \right\},\ \mathbb{A}_{\mathbf{Q}}(\mathbf{V}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}_2).$$

二つのルールは共通の述語 $\mathbb{A}_{\mathbf{R}}$ と共通でない述語 $\mathbb{A}_{\mathbf{P}}, \mathbb{A}_{\mathbf{Q}}$ を持つ．二つのルールが任意のルールをあらわすため $\mathbb{A}_{\mathbf{P}}, \mathbb{A}_{\mathbf{Q}}, \mathbb{A}_{\mathbf{R}}$ には空集合を含む任意の述語の集合が許される．ただし，ルールヘッドが空にならないよう $(\mathbb{A}_{\mathbf{P}} \cup \mathbb{A}_{\mathbf{R}})$ および $(\mathbb{A}_{\mathbf{Q}} \cup \mathbb{A}_{\mathbf{R}})$ は空でないとする．排他的マージ方法によって生成されるルールは以下の形をしている．

$$r_E : \mathbb{A}_{\mathbf{P}}(\mathbf{X}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{Y}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{Z}),\ \{\mathrm{cond}_E\}\ ==>\ \{\mathrm{exec}_E\},\ \mathbb{A}_{\mathbf{P}}(\mathbf{U}) \cup \mathbb{A}_{\mathbf{Q}}(\mathbf{V}) \cup \mathbb{A}_{\mathbf{R}}(\mathbf{W}). \tag{5.27}$$

cond_E と exec_E の決め方は「排他的マージの説明」(§5.6.3) と同様の考え方に基づいて議論できる．すなわち条件部 $\mathrm{cond}_1, \mathrm{cond}_2$ の真偽の組み合わせに対して合成ルール r_E が r_{31}, r_{32} と等価な作用をすればよい．

cond_E と exec_E は以下のように定義される．

■exec_E： §5.6.3 と同じ議論により，実行部 exec_E は下記の 3 条件を満たすような代入 ρ_{exec} を発生させる手続きであればよい．

1. $\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \neg\mathrm{cond}_2(\mathbf{Y},\mathbf{Z})$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{F}_p(\mathbf{X},\mathbf{Z}),\ \mathbf{V}/\mathbf{Y},\ \mathbf{W}/\mathbf{F}_r(\mathbf{X},\mathbf{Z}) \right\}$
2. $\neg\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z})$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{X},\ \mathbf{V}/\mathbf{G}_q(\mathbf{Y},\mathbf{Z}),\ \mathbf{W}/\mathbf{G}_r(\mathbf{Y},\mathbf{Z}) \right\}$
3. $\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z})$
   $\Rightarrow\ \rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{F}_p(\mathbf{X},\mathbf{Z}),\ \mathbf{V}/\mathbf{Y},\ \mathbf{W}/\mathbf{F}_r(\mathbf{X},\mathbf{Z}) \right\}$
   または
   $\rho_{\mathrm{exec}} = \left\{ \mathbf{U}/\mathbf{X},\ \mathbf{V}/\mathbf{G}_q(\mathbf{Y},\mathbf{Z}),\ \mathbf{W}/\mathbf{G}_r(\mathbf{Y},\mathbf{Z}) \right\}$
4. 上記以外 $\Rightarrow$ 実行部は任意

条件 3. の最初の代入を採用すれば ルール r_{31} 優先の合成ルールになり，二番目を採用すれば ルール r_{32} 優先の合成ルールになる．また，$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ のコンテクストに応じて採用する代入を変えても構わない．

■**cond**$_E$： 上記のうちルール r_S が適用されるべきケースは 1. 2. 3. の 3 通りである．cond_E はこれらの前提条件の論理和として定義すればよい．

$$\begin{aligned}\mathrm{cond}_E \stackrel{\mathrm{def}}{=} &(\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \neg\mathrm{cond}_2(\mathbf{Y},\mathbf{Z})) \\ &\vee (\neg\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z})) \\ &\vee (\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \wedge \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))\end{aligned}$$

これは $(\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))$ に等しい．

■**実装例**:

$\mathrm{cond}_E : (\mathrm{cond}_1(\mathbf{X},\mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y},\mathbf{Z}))$
exec_E :
if $\{\mathrm{cond}_1(\mathbf{X},\mathbf{Z})\}$ then
$\{\ \mathbf{U} := \mathbf{F}_p(\mathbf{X},\mathbf{Z});\ \mathbf{V} := \mathbf{Y};\ \mathbf{W} := \mathbf{F}_r(\mathbf{X},\mathbf{Z});\ \}$
else
$\{\ \mathbf{U} := \mathbf{X};\ \mathbf{V} := \mathbf{G}_q(\mathbf{Y},\mathbf{Z});\ \mathbf{W} := \mathbf{G}_r(\mathbf{Y},\mathbf{Z});\ \};$

5.8 ルールのマージの正当性

マージ手法の正当性は，マージで得られた合成ルールが**等価変換 (ET) ルール** (式 (2.12)) であることを示すことで保証される．合成ルールによって書き換えられた確定節が質問の “意味” を保存しているならば，その合成ルールは ET ルールである．

具体的には，合成ルールを適用して得られる確定節が合成前のオリジナルルールでも得られることを示せばよい．オリジナルルールは ET ルールなので，オリジナルルールによって書き換えられた確定節は質問の意味を保存している．

証明は省略するが,「並列的マージ」,「シーケンシャルなマージ」,「排他的マージ」のいずれも得られる合成ルールは ET ルールになっていることが確認できる．

第 6 章

ET ルール変換フレームワーク

6.1　回路生成フレームワークと生成手順

ここでは理論的に示される大きなフレームワークと，そのフレームワークに則って実際の変換を行う流れを説明する．

6.1.1　フレームワーク

フレームワークを 図 6.1 に示す．この図の中で Primitive ET rules はターゲットのデバイス/システムに適したルール (Dedicated ET rules) へ変換される．このような変換方法は 5.2 章で議論した代数系変換の理論を利用して開発される．そして，開発された変換方法はデータベース化しておく (Transformation Strategy).

ET ルールの変換はいくつかのフェーズに分けても構わない．各フェーズで ET ルール

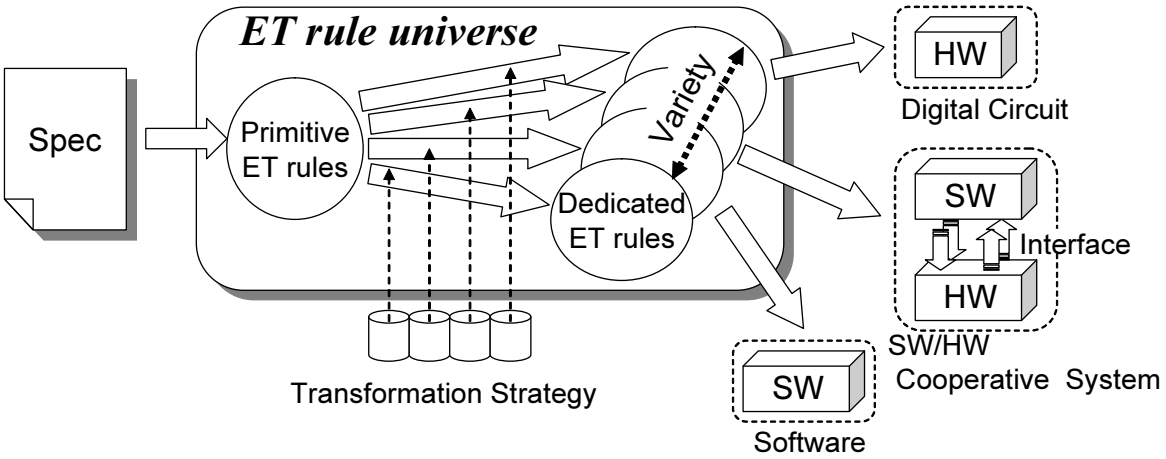


図 6.1　Framework Synthesizing Digital Circuit

が少しずつターゲットに近づいてゆくように変換することができる．

$$\text{宣言的仕様} \longrightarrow \text{ET ルール } 0 \longrightarrow \cdots \longrightarrow \text{ET ルール } N \longrightarrow \text{ターゲット}$$

ここで $N \geq 0$ である．ET ルールの変換は ET ルールの空間 (*ET rule universe*) の中で行われ，代数系が変わっても変換された結果が ET ルールであることは保証される．

このフレームワークの特徴

1. 宣言的仕様から直接回路を生成するのではなく，ET ルールを経由して回路を生成する．
2. 宣言的仕様から ET ルール 0 を作る先行研究 [7–12] があり，そのサポートが受けられる．
3. ET ルール $(i-1)$ から ET ルール i の変換と，ET ルール N からターゲットへの変換は本論文で紹介した代数系変換理論がサポートする．
4. ET ルール 0 から ET ルール N は ET ルールの空間内でターゲットに応じて任意の数に分割できる．

最終出力はターゲットのプラットフォームにあわせたソース，バイナリ，ネットリストなどである．HW 用に生成されるものは VHDL/Verilog 等のソースやネットリストであり，SW 用に生成されるものは C++/Java 等のソースや実行バイナリである．ターゲットが ET インタプリタの場合は N=0 でも構わないし，計算効率のよい ET ルール集合に変換することもできる．

6.1.2 回路記述のためのビルトイン述語

本研究は，確定節で記述された宣言的仕様から回路を生成することを目的としている．宣言的仕様は確定節によって記述され，確定節はアトムを用いて記述される．そのためには宣言的仕様を記述するのに十分な有限個のビルトイン述語を準備しておく必要がある．

本手法でビルトイン述語として扱えるものは組み合わせ回路 (§1.3.4) だけで実現できるものである (正確には，1 クロック周期以内に計算結果が出る回路)．この制約を満たすものならば，どのようなビルトイン述語も可能である．したがってビルトイン述語にできるものは，その時代のテクノロジーに大きく依存する．組み合わせ回路で実現できないものは確定節で宣言的仕様を定義し，本研究の手法によって順序回路 (§1.3.4) として生成すればよい．

> 組み合わせ回路で実現できるものは一般に関数である．例えば，論理演算, ビット演算, 整数の比較, 加減算, 乗算などは組み合わせ回路で実現可能である．除算も関数であるが，本書を執筆している時点では回路の規模が実用的ではないためビルトイン述語として扱わない．

6.2 ET ルールの変換

前章のフレームワークは理論的な側面から大きな流れを示したものである．しかしターゲットに応じて細かい理論や技術などが必要になる．なぜならターゲットのプラットフォームによって，使えるエレメントや資源や実行方法などが異なるからである．例えばHW へ変換する場合にはポインタやスタックは使えないが並列計算は可能である．一方SW の場合には並列計算はできないがポインタやスタックは使える．

6.2.1 ポリシーとストラテジー

まず最初に必要なのはどのように節をレジスタへマッピングするのか (ポリシー) と，そのポリシーを実現するためにどのような手順で ET ルールを変換してゆくのか (ストラテジー) の二つである．

ポリシーの作り方やストラテジーの立て方は用途に応じて様々考えられるため，今後の研究課題である．本論文では，実際に ET ルールから回路を生成できることを示すため，次のように定めたポリシーとストラテジーを用いる．

ポリシー

ここでは，ET で質問節の計算を行ったときに節中に現れる全てのアトムに対して，その引数に固有のレジスタを割り当てることにする．回路のレジスタは有限個しかないため，節の長さや出現するアトムの種類が有限のものだけを扱う．

まずナイーブに節の引数をレジスタに割り当ててから，レジスタを共有できる引数を見つけて冗長性を排除する．

レジスタ共有の抽出によって一部の変数への代入が解消されることがある (共有変数を持った引数を同じレジスタに共有)．先に共有レジスタの抽出を行えば，変数代入解消作業の負担が減る．

並列計算アトムは宣言的仕様に近いレベルで抽出した方が効率的に見つけられる．そこで最初に並列計算アトムを抽出する．

非決定性の解消は計算パスを限定する．限定の仕方によっては変数への代入の解消や共有レジスタの抽出に影響を与え効率が悪くなる可能性がある．そこで一番最後に行う．

ストラテジー

上のポリシーに沿って立てたストラテジーを示す．

1. 宣言的仕様に近いレベルで並列計算可能アトムを抽出
2. 節から固定長の節へ変換 (代数系変換)
3. 節の中でレジスタを共有できる引数を抽出
4. ルールの変数代入の解消 (代数系変換)

$$\mathbb{D} = \left\{ \begin{array}{l} \mathsf{main}(N, M, Z) \leftarrow \mathsf{gcd}(N, M, X), \mathsf{fact}(X, 1, Z). \\ \mathsf{gcd}(N, M, X) \leftarrow \mathsf{greater}(N, M), \mathsf{gcd}(M, N, X). \\ \mathsf{gcd}(N, M, X) \leftarrow \mathsf{lesseq}(N, M), \mathsf{neq}(N, 0), \mathsf{sub}(M, N, M1), \mathsf{gcd}(N, M1, X). \\ \mathsf{gcd}(0, M, M) \leftarrow . \end{array} \right\}$$

図 6.2　GCD 背景知識 $\mathbb{D}$

$$\mathbb{Q} = \left\{ \mathsf{ans}(Z) \leftarrow \mathsf{main}(n, m, Z) \;\middle|\; n, m \in \mathbb{N},\ Z \in \mathbb{V} \right\}$$

where $\mathbb{N}$ is a set of natural numbers, $\mathbb{V}$ is a set of logic variables.

図 6.3　GCD 質問 $\mathbb{Q}$

5. ルール適用の非決定性を解消する (代数系変換)
6. FSM の回路へマッピングする (代数系変換)

6.3　GCD を用いた例題

6.3.1　GCD の宣言的仕様

宣言的仕様は背景知識 $\mathbb{D}$ と 質問 $\mathbb{Q}$ のペア $\langle \mathbb{D}, \mathbb{Q} \rangle$ で与えられる．$\mathbb{D}$ は GCD を計算するために必要な背景知識を確定節の集合によって表したものである．GCD の背景知識 $\mathbb{D}$ は 図 6.2 の確定節集合で与えられる．ただし greater, lesseq, neq, sub, mul は既知の**ビルトインアトム**とし下記に示された関係を表しているものとする．

$$\begin{aligned} \mathsf{greater}(A, B) &: A > B \\ \mathsf{lesseq}(A, B) &: A \leq B \\ \mathsf{neq}(A, B) &: A \neq B \\ \mathsf{sub}(A, B, C) &: C = A - B \\ \mathsf{mul}(A, B, C) &: C = A \times B \end{aligned}$$

そのほかのアトムはユーザー定義アトムであり，main は回路を呼び出すための上位モジュール，$\mathsf{gcd}(N, M, X)$ は $X = \mathrm{GCD}(N, M)$ という関係を意図した述語である．

質問 $\mathbb{Q}$ はプログラムが対応すべき問題の範囲を明確にする．例えば任意の二つの自然数 n, m を与えそれらの $\mathrm{GCD}(n, m)$ を計算する質問 $\mathbb{Q}$ は図 6.3 の確定節集合で与えられる．この質問節から，入力が n, m で出力が Z であることが分かる．もし 自然数 n, z を与えて $z = \mathrm{GCD}(n, M)$ を満たすような M を計算したいなら，下記ような質問の定義も可能である．

/* Call rule */
r_{g1} : main(N, M, Z) ==> gcd(N, M, X), fact$(X, 1, Z)$.

/* GCD rule */
r_{g2} : gcd(N, M, X), { $(N$ is-ground$) \wedge (M$ is-ground$) \wedge (N > M)$ }
==> gcd(M, N, X).

r_{g3} : gcd(N, M, X), { $(N$ is-ground$) \wedge (M$ is-ground$) \wedge (N \leq M) \wedge (N \neq 0)$ }
==> { $M1 := M - N$ }, gcd$(N, M1, X)$.

r_{g4} : gcd$(0, M, X)$ ==> { $X := M$ }.

図 6.4　GCD ET ルール

$$\mathbb{Q}' = \left\{ \mathsf{ans}(M) \leftarrow \mathsf{main}(n, M, z) \;\middle|\; n, z \in \mathbb{N},\ M \in \mathbb{V} \right\}$$

ただし，この計算は答えが一意に決まらないため，今回は例題として用いない．

6.3.2　ルール生成

宣言的仕様 $\langle \mathbb{D}, \mathbb{Q} \rangle$ から ET ルールを生成する方法には，搾り出し法，メタ計算による方法，等価論理式による方法などがある [7–12]．本研究は ET ルール生成の研究ではないので生成方法については省略する．ここでは図 6.4 の ET ルールが得られたと仮定し，話を進める．この ET ルールは，そのまま実行することが可能である．実行例として main$(51, 27, X)$ を ET インタプリタ (ETI [20]) で計算した過程を図 6.5 に示す．最後に得られた単位節は答えが 3 に等しいことを示している．実際，GCD$(51, 27) = 3$ である．

6.3.3　ルール変換

図 6.4 を直ちに回路にすることはできない．回路化するには 4 章で示したように FSM と等価な動作をする ET ルールに変換しなければならない．そのために図 6.4 のルール集合を下記の制約を満たす等価な ET ルール集合に変換する．

1. ルールのヘッドアトムの変数へ代入をおこなわない．
2. ルールのヘッドアトムとボディアトムが同じ形であること．つまり引数だけが異なる同種のアトム集合になっていること．
3. 条件部と実行部は組み合わせ回路だけで実現できるものとする．

```
(main 51 27 *A)
-------------------------N execution ---------------------
============================
(ans (main 51 27 *A))<-(gcd 51 27 *A).
============================
(ans (main 51 27 *A))<-(gcd 27 51 *A).
============================
(ans (main 51 27 *A))<-(gcd 27 24 *A).
============================
(ans (main 51 27 *A))<-(gcd 24 27 *A).
============================
(ans (main 51 27 *A))<-(gcd 24 3 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 24 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 21 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 18 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 15 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 12 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 9 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 6 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 3 *A).
============================
(ans (main 51 27 *A))<-(gcd 3 0 *A).
============================
(ans (main 51 27 *A))<-(gcd 0 3 *A).
============================
(ans (main 51 27 3))<-.
------------------------------------------------------------
(ans (main 51 27 3))<-.
```

図 6.5　実行例

```
main(busy, N, M, Z), gcd(free, _, _, _)
    ==> main(free, _, _, _), gcd(busy, N, M, Z).

gcd(busy, N, M, X), {(N is ground) ∧ (M is ground) ∧ (N > M)}
    ==> gcd(busy, M, N, X).

gcd(busy, N, M, X), {(N is ground) ∧ (M is ground) ∧ (N ≤ M) ∧ (N ≠ 0)}
    ==> {M1 := M − N}, gcd(busy, N, M1, X).

gcd(busy, 0, M, X) ==> { X := M }, gcd(stop, _, _).
```

図 6.6　GCD ヘッド–ボディ統一

制約 1 の理由は共有変数によって他のアトムの引数に値を伝播させるためにポインタが必要になるからである．値の伝播は共有変数を用いずにマルチヘッドルールを用いてアトムを replace して実現する．

制約 2 の理由は動的にアトムすなわちメモリを生成，消滅しないためである．Flip Flop を動的に生成／消滅することはできない．また，この制約によりルールは単純な末尾再帰となりループで計算できる．

制約 3 を満たす計算には，論理演算，数値の比較，加減算，乗算などがある．除算や剰余は組み合わせ回路だけでは実現できない．この制約を満たさない計算はユーザー定義アトムを定義し，ルールによってアルゴリズムを記述する．

上記の制約によって計算が単純な FSM を用いて実現でき，コンパクトで効率のよい回路が生成できる．ET ルールにこのような制約を課しても宣言的仕様は影響を受けない．なぜなら ET では宣言的仕様とアルゴリズムを分離しているからである．実装を気にせず宣言的仕様を記述し，回路生成に都合のよい ET ルールを生成すればよい．

ルールヘッドとルールボディを同じ形にする

ルールヘッドとルールボディを同じ形にする．その方法は単純である．アトムの引数に，そのアトムの存在を表すフラグを導入する．例えば

$$p(X) \texttt{==>} r(Y).$$

というルールの場合

$$p(\text{busy}, X), r(\text{free}, \_) \texttt{==>} p(\text{free}, \_), r(\text{busy}, Y).$$

のように変形するとヘッドとボディを同じ形にできる．引数に追加されたフラグ free は，そのアトムが空アトムに等しいことを表す．busy フラグは，そのアトムが空アトムではないことを表す．また，計算の終了を表す stop も導入する．

r_{call}: main(N, M, X), gcd(free, _ , _ , _), { cond$_{call}$ } ==> main(N, M, X), gcd(busy, N, M, X).
r_{get} : main$(N, M,$ _), gcd(stop, $N1, M1, X$) ==> main(N, M, X), gcd(free, $N1, M1, X$).
r_{m1} : gcd(busy, $0, M, X$) ==> gcd(stop, $0, M, M$).
r_{m2} : gcd(busy, N, M, X), $\{N > M\}$ ==> gcd(busy, M, N, X).
r_{m3} : gcd(busy, N, M, X), $\{(N \leq M) \wedge (N \neq 0)\}$ ==> $\{M1 := M - N\}$, gcd(busy, $N, M1, X$).

図 6.7　GCD 実用回路のルール

この変換を 図 6.4 のルール集合に施すと図 6.6 のルール集合が得られる．計算状態を表すフラグが gcd アトムの第一引数の前に追加されている．さらに r_{m1} にはボディアトム gcd$(\text{stop}, 0, M, M)$ が追加されている．このアトムだけが stop フラグを持っており，それ以外の gcd アトムは busy フラグを持っている．stop フラグを持った gcd アトムを計算終了とみなし，busy フラグを述語名の一部と考えると 図 6.6 は図 6.4 のルールと全く同じであることが分かる．

次に入出力を追加する．入出力の情報は質問 $\mathbf{Q}$ の形から与えられる．すなわち m, n が入力であり X が出力である．これらの入力値をセットする機構を追加する必要がある．gcd の計算をする回路はモジュールとして呼び出されるので gcd モジュールを呼び出すルールとこのモジュールから結果を受け取るルールを記述する．

図 6.7 は呼び出しをするルール r_{call} と結果を受け取るルール r_{get} を追加したものである．r_{call} ルールによって main アトムから gcd アトムへ入力が渡され，r_{get} ルールによって gcd アトムから main アトムへ結果が渡される．つまり main アトムが gcd モジュールを呼び出す上位の回路に相当している．r_{call} の条件部 cond$_{call}$ は計算開始のトリガーであり通常の計算では $\{X$ is Variable$\}$ のような条件が入る．この例題では main アトムは gcd モジュールを呼び出すインターフェースとして回路化されるので，cond$_{call}$ には特別な計算開始信号が割り当てられる．

6.3.4　ルールのマージ

ルールのマージによって全てのルールを 1 つの等価な ET ルールに変換する．その結果図 6.7 のルール集合は一つのルールに変換される．この章では，ET ルールをマージする過程を具体的に説明する．

図 6.7 のルールのマージには排他的なマージとシーケンシャルなマージを用いることができる．しかし全てのルールが共通の述語 gcd をヘッドに持つため並列的ルールのマージは利用できない．まず最初に r_{m2} と r_{m3} から排他的マージ (p.86) によって等価なルールを生成する．まず，これら二つのルールから一般化された排他的マージの方法のパラメータテーブルを作成する (図 6.8)．ただし，第一引数 busy は共通なので述語名の一部とみなす．$\mathbb{A}_{\mathbf{P}}$ と $\mathbb{A}_{\mathbf{Q}}$ が空なので生成されるルールは形式的に下記の形であらわせる

$$
\begin{aligned}
\mathbb{A}_{\mathbf{P}} &= \mathbb{A}_{\mathbf{Q}} = \emptyset \\
\mathbf{X} &= \mathbf{Y} = \langle\rangle \\
\mathbb{A}_{\mathbf{R}} &= \{\mathsf{gcd}\} \\
\mathbf{Z} &= \mathbf{Z}_1 = \mathbf{Z}_2 = \langle N, M, X\rangle \\
\mathbf{U} &= \mathbf{V} = \langle\rangle \\
\mathbf{W} &= \langle U, V, W\rangle \\
\mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) &= (N > M) \\
\mathrm{cond}_2(\mathbf{Y}, \mathbf{Z}) &= (N \le M) \wedge (N \ne 0) \\
\mathbf{F}_p &= \mathbf{G}_q = \langle\rangle \\
\mathbf{F}_r(\mathbf{X}, \mathbf{Z}_1) &= \langle M, N, X\rangle \\
\mathbf{G}_r(\mathbf{Y}, \mathbf{Z}_2) &= \langle N, M - N, X\rangle
\end{aligned}
$$

図 6.8　排他的合成 parameter (r_{m2}, r_{m3})

$$
\begin{aligned}
\mathrm{cond}_{m23} &: (\mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) \parallel \mathrm{cond}_2(\mathbf{Y}, \mathbf{Z})) \\
\mathrm{exec}_{m23} &: \mathbf{W} = \delta(\mathrm{cond}_1) \times \mathbf{F}_r(\mathbf{X}, \mathbf{Z}) + \delta(\neg\mathrm{cond}_1) \times \mathbf{G}_r(\mathbf{Y}, \mathbf{Z})
\end{aligned}
$$

ただし $\delta(x)$ は論理式 x の値が真のとき 1, 偽のとき 0 となるスカラ関数である.

図 6.9　排他的合成 計算式 (r_{m2}, r_{m3})

$$
\begin{aligned}
r_{m23} : \mathsf{gcd}(\mathrm{busy}, N, M, X), \{\mathrm{cond}_{m23}\} \\
\text{==> } \{\ \mathrm{exec}_{m23}\ \},\ \mathsf{gcd}(\mathrm{busy}, U, V, W).
\end{aligned}
$$

排他的合成の条件部と実行部を求める式は 図 6.9 のとおり. この式から求めた新たな条件部は下記のとおり.

$$
\begin{aligned}
\mathrm{cond}_{m23} &= \mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y}, \mathbf{Z}) \\
&= (N > M) \vee \Big((N \le M) \wedge (N \ne 0)\Big) \\
&= (N > M) \vee \Big(\neg(N > M) \wedge (N \ne 0)\Big) \\
&= (N > M) \vee (N \ne 0)
\end{aligned}
$$

r_{m23}: gcd(busy, N, M, X), $\{(N > M) \vee (N \neq 0)\}$
==> { $U := f_1(N, M)$; $V := f_2(N, M)$; },
gcd(busy, U, V, X).
where
$f_1(N, M) = \delta(N > M) \times M + \delta(\neg(N > M)) \times N$
$f_2(N, M) = \delta(N > M) \times N + \delta(\neg(N > M)) \times (M - N)$

図 6.10　排他的合成後のルール

同様に新たな実行部は以下のように求まる

$$
\begin{aligned}
U &:= \delta(N > M) \times M + \delta(\neg(N > M)) \times N; \\
V &:= \delta(N > M) \times N + \delta(\neg(N > M)) \times (M - N); \\
W &:= X;
\end{aligned}
$$

この結果，得られるルールは 図 6.10 のとおり．ルール r_{m23} ひとつで r_{m2} と r_{m3} の両方を置き換えることができる．

続いて r_{m23} と r_{m1} からシーケンシャル合成によって等価なルールを合成する．r_{m1} は計算終了時に 1 回だけ適用されるルールであり必ず r_{m23} より後に適用される．従ってルールの適用順序は r_{m23}, r_{m1} の順となる．また今回は gcd の第一引数 (フラグ) が異なるのでフラグも条件部へ入れる．これら二つのルールから一般化されたシーケンシャル合成の方法のパラメータテーブルを作成する (図 6.11)．$\mathbb{A}_{\mathbf{P}} = \mathbb{A}_{\mathbf{Q}} = \emptyset$ なので生成されるルールは形式的に下記の形に表せる

$$
\begin{aligned}
r_{m123} : \mathsf{gcd}(F, N, M, X), \{\mathrm{cond}_{m123}\} \\
\Longrightarrow \{\, \mathrm{exec}_{m123} \,\},\ \mathsf{gcd}(K, U, V, W).
\end{aligned}
$$

シーケンシャル合成の条件部と実行部を求める式は図 6.12 のとおり．この式から求めた新たな条件部は下記のとおりである．

$$
\begin{aligned}
\mathrm{cond}_{m123} &= \mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) \vee \mathrm{cond}_2(\mathbf{Y}, \mathbf{Z}) \\
&= \Big((F = \mathrm{busy}) \wedge ((N > M) \vee (N \neq 0))\Big) \vee \Big((F = \mathrm{busy}) \wedge (N = 0)\Big) \\
&= (F = \mathrm{busy}) \wedge \Big((N > M) \vee (N \neq 0) \vee (N = 0)\Big) \\
&= (F = \mathrm{busy})
\end{aligned}
$$

同様に新たな実行部は下記のとおりである．ただしルールが適用されるとき条件部が必ず真なので実行部では $(F = \mathrm{busy})$ を仮定して構わない．下記の実行部はこの仮定を用いて

$$
\begin{aligned}
\mathbb{A}_{\mathbf{P}} &= \mathbb{A}_{\mathbf{Q}} = \emptyset \\
\mathbb{A}_{\mathbf{R}} &= \{\mathsf{gcd}\} \\
\mathbf{X} &= \mathbf{Y} = \langle\rangle \\
\mathbf{Z} &= \mathbf{Z}_1 = \mathbf{Z}_2 = \langle F, N, M, X\rangle \\
\mathbf{U} &= \mathbf{V} = \langle\rangle \\
\mathbf{W} &= \langle K, U, V, W\rangle \\
\mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) &= (F = \mathrm{busy}) \wedge \Big((N > M) \vee (N \neq 0)\Big) \\
\mathrm{cond}_2(\mathbf{Y}, \mathbf{Z}) &= (F = \mathrm{busy}) \wedge (N = 0) \\
\mathbf{F}_p &= \mathbf{G}_q = \langle\rangle \\
\mathbf{F}_r(\mathbf{X}, \mathbf{Z}_1) &= \langle \mathrm{busy}, f_1(N, M, X), f_2(N, M, X), X\rangle \\
\mathbf{G}_r(\mathbf{Y}, \mathbf{Z}_2) &= \langle \mathrm{stop}, 0, M, M\rangle
\end{aligned}
$$

図 6.11　シーケンシャル合成 parameter (r_{m23}, r_{m1})

$$
\begin{aligned}
\mathrm{cond}_{m123}: \ & (\mathrm{cond}_1(\mathbf{X}, \mathbf{Z}) \parallel \mathrm{cond}_2(\mathbf{Y}, \mathbf{Z})) \\
\mathrm{exec}_{m123}: \ & \mathbf{T} := \delta(\mathrm{cond}_1(\mathbf{X}, \mathbf{Z})) \times \mathbf{F}_r(\mathbf{X}, \mathbf{Z}) + \delta(\neg\mathrm{cond}_1(\mathbf{X}, \mathbf{Z})) \times \mathbf{Z}; \\
& \mathbf{V} := \delta(\mathrm{cond}_2(\mathbf{Y}, \mathbf{T})) \times \mathbf{G}_q(\mathbf{Y}, \mathbf{T}) + \delta(\neg\mathrm{cond}_2(\mathbf{Y}, \mathbf{T})) \times \mathbf{Y}; \\
& \mathbf{W} := \delta(\mathrm{cond}_2(\mathbf{Y}, \mathbf{T})) \times \mathbf{G}_r(\mathbf{Y}, \mathbf{T}) + \delta(\neg\mathrm{cond}_2(\mathbf{Y}, \mathbf{T})) \times \mathbf{T};
\end{aligned}
$$

図 6.12　シーケンシャル合成 計算式 (r_{m23}, r_{m1})

簡略化されている.

$$
\begin{aligned}
T_1 &:= \delta((N > M) \vee (N \neq 0)) \times f_1(N, M) + \delta(\neg((N > M) \vee (N \neq 0))) \times N; \\
T_1 &:= \delta((N > M) \vee (N \neq 0)) \times f_1(N, M) + \delta(\neg((N > M) \vee (N \neq 0))) \times N; \\
T_2 &:= \delta((N > M) \vee (N \neq 0)) \times f_2(N, M) + \delta(\neg((N > M) \vee (N \neq 0))) \times M; \\
T_3 &:= X; \\
G &:= \delta((T_1 = 0) \wedge (F = 0)) \times 1 + \delta(\neg((T_1 = 0) \wedge (F = 0))) \times F; \\
H &:= \delta((T_1 = 0) \wedge (F = 0)) \times M + \delta(\neg((T_1 = 0) \wedge (F = 0))) \times Y; \\
U &:= \delta((T_1 = 0) \wedge (F = 0)) \times N + \delta(\neg((T_1 = 0) \wedge (F = 0))) \times T_1; \\
V &:= \delta((T_1 = 0) \wedge (F = 0)) \times M + \delta(\neg((T_1 = 0) \wedge (F = 0))) \times T_2; \\
W &:= \delta((T_1 = 0) \wedge (F = 0)) \times Y + \delta(\neg((T_1 = 0) \wedge (F = 0))) \times T_3;
\end{aligned}
$$

この結果，得られる合成ルールは 図 6.13 である.

$$
\begin{aligned}
&r_{m123}\colon \mathsf{gcd}(F,N,M,X),\{F = \mathrm{busy}\}\\
&\quad ==>\ \{\, K := g_1(F,N,M,X);\ U := g_2(F,N,M,X);\ V := g_3(F,N,M,X);\\
&\qquad\quad W := g_4(F,N,M,X);\,\},\\
&\qquad\quad \mathsf{gcd}(K,U,V,X),\mathsf{ret}(G,H).
\end{aligned}
$$

where

$$
\begin{aligned}
g_1(N,M,X,F,Y) = {}& \delta((F=\mathrm{busy})(T_1=0)\wedge(F=0))\times N\\
&+\ \delta(\neg((T_1=0)\wedge(F=0)))\times T_1;\\
g_2(N,M,X,F,Y) = {}& \delta((T_1=0)\wedge(F=0))\times M\\
&+\ \delta(\neg((T_1=0)\wedge(F=0)))\times T_2;\\
g_3(N,M,X,F,Y) = {}& \delta((T_1=0)\wedge(F=0))\times Y\\
&+\ \delta(\neg((T_1=0)\wedge(F=0)))\times X;\\
g_4(N,M,X,F,Y) = {}& \delta((T_1=0)\wedge(F=0))\times 1\\
&+\ \delta(\neg((T_1=0)\wedge(F=0)))\times F;\\
g_5(N,M,X,F,Y) = {}& \delta((T_1=0)\wedge(F=0))\times M\\
&+\ \delta(\neg((T_1=0)\wedge(F=0)))\times Y;\\
T_1 = {}& \delta((N>M)\vee(N\neq 0))\times f_1(N,M,X)\\
&+\ \delta(\neg((N>M)\vee(N\neq 0)))\times N;\\
T_2 = {}& \delta((N>M)\vee(N\neq 0))\times f_2(N,M,X)\\
&+\ \delta(\neg((N>M)\vee(N\neq 0)))\times M;
\end{aligned}
$$

図 6.13　シーケンシャル合成結果 (r_{m23}, r_{m1})

マージされた最終的なルール

以上のように，図 6.7 の r_{m1}, r_{m2}, r_{m3} の 3 つのルールがマージされて図 6.13 に示される r_{m123} のルールになる．同様に，ルール r_{m123} に図 6.7 の r_{call} と r_{get} をマージすると，最終的に図 6.14 で示される形式のルールが一つだけとなる．このルールは図 6.7 の 5 つのルールと全く等価である．そして，この形のルールは 4 章で紹介した有限状態マシンの動作をあらわすルールになっており，容易にデジタル回路へ変換できることが分かる．

r_m: $\mathsf{main}(A, B, C), \mathsf{gcd}(F, N, M, X), \{\mathrm{cond}_{rm}\}$
==> { $H := f_1(A, B, C, F, N, M, X)$; $K := f_2(A, B, C, F, N, M, X)$;
$U := f_3(A, B, C, F, N, M, X)$; $V := f_4(A, B, C, F, N, M, X)$;
$W := f_5(A, B, C, F, N, M, X)$; },
$\mathsf{main}(A, B, H), \mathsf{gcd}(K, U, V, W)$.

図 6.14　目標とする GCD 合成 ET ルールの形

第 7 章

協調計算システムへの応用

計算量が多く並列計算できる部分を多く含む問題はソフトウエアとハードウエアの協調計算によって効率よく解くことができる．制約充足問題などの制約充足問題は計算量が多く並列計算可能な部分が多いため HW/SW 協調計算を利用すると非常に効率よく解けることが期待できる．

我々は制約充足問題を解くための協調計算システムを開発する枠組みを提案する．この枠組みによって正しい協調計算システムを簡単に実現することができる．この枠組みの中で用いられるシステム記述言語は ET ルール [3, 5, 8, 9] である．一般に，協調システムを記述するには SystemC[28] や SystemVerilog[29] が用いられることが多いが，これらの言語は制約充足問題の記述には向いていない．ET は Lisp や PROLOG などと同じく制約充足問題の記述に適した言語である．さらに ET には Lisp や PROLOG には無い優れた特徴がある．その特徴を用いると問題の実行を分割して分散並列協調システムへ容易にマッピングできる．これについては §7.3 で示す．

本章の構成は以下のとおりである．§7.1 で我々が実現しようとしているシステムの概要を説明する．§7.2 で例題を用いながら ET の計算の特徴を示す．§7.3 で ET の並列計算を示し，ET を用いてシステムを記述利点を議論する．§7.4 では ET による分散協調計算を実現するために開発したサーバーシステムのアーキテクチャや性能評価結果を示す．

7.1 システムの概要

一般に分散計算や並列計算は PC クラスタなどのように複数の PC をネットワークで接続して利用する．この形態では各サーバー PC の CPU リソースをクライアント PC が利用する構造になっている．我々はサーバー PC に回路によるアクセラレーション機能を持たせた図 7.1 に示すような分散並列協調システムを考えた [56, 57]．このシステムは ET で記述された問題をサーバーにインストールされた回路ボードで計算をアクセラレーションする．ユーザーはサーバーコンピュータの CPU リソースと回路リソースの両方を利用して高効率な計算を実現できる．

このシステムでは回路が FPGA によって動的に再構成できる．サーバーにインストール

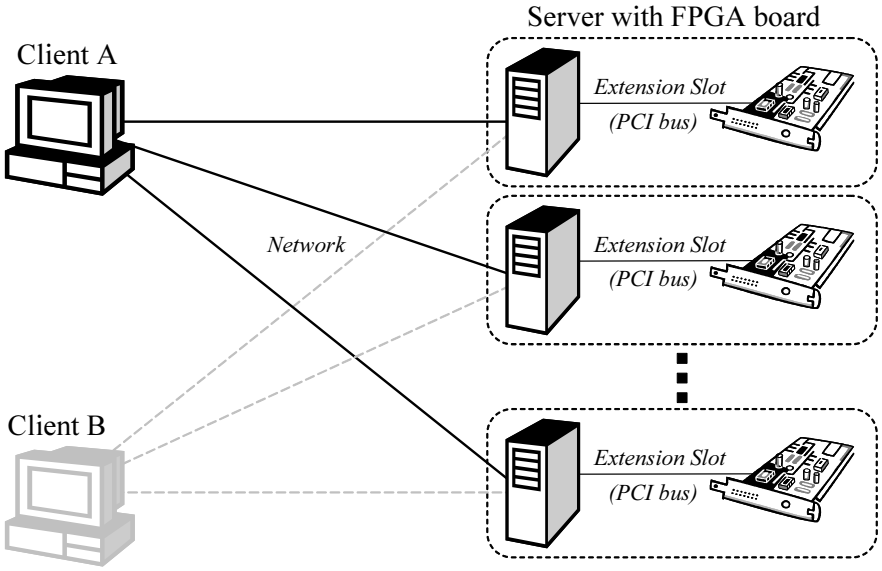


図 7.1　分散並列協調システム

される回路ボードは HwModule Board[58, 59] を利用する．このボード上には 3 つのユーザー FPGA があり，この FPGA は PCI バスを経由してサーバー上のプログラムからアクセスされる．サーバーがリモートクライアントから回路情報を受け取って FPGA を再構成する使用法も想定している．回路リソースに空きがあれば複数のクライアントが同時に異なる回路をロードして使うこともできる．これにより回路やサーバー資源を有効に使うことができる．

7.1.1　なぜ ET？

このような協調計算システムを使う上で問題となるのが如何にして問題を記述し，如何にして機能分割を行うかということである．SystemC や SystemVerilog で制約充足問題を記述するのは難しいし，協調計算のパフォーマンスは機能分割に依存する．

一般に並列度の高い計算ほど回路化の恩恵が得られるが，並列性を発見するのは難しく，並列計算による結果の正当性を保証することも難しい．

回路化できるかどうかも重要な問題である．任意に機能分割してしまうと回路化できない場合がある．回路化に適した分割を見つけるためには様々な粒度と並列度で分割点を発見できる枠組みが重要である．

ET を選ぶ理由には以下のようなものがあげられる

- 制約充足問題記述に向いている
- 正当性の理論がある (赤間 *et al.*).
- 宣言的仕様・SW・HW を記述できる．

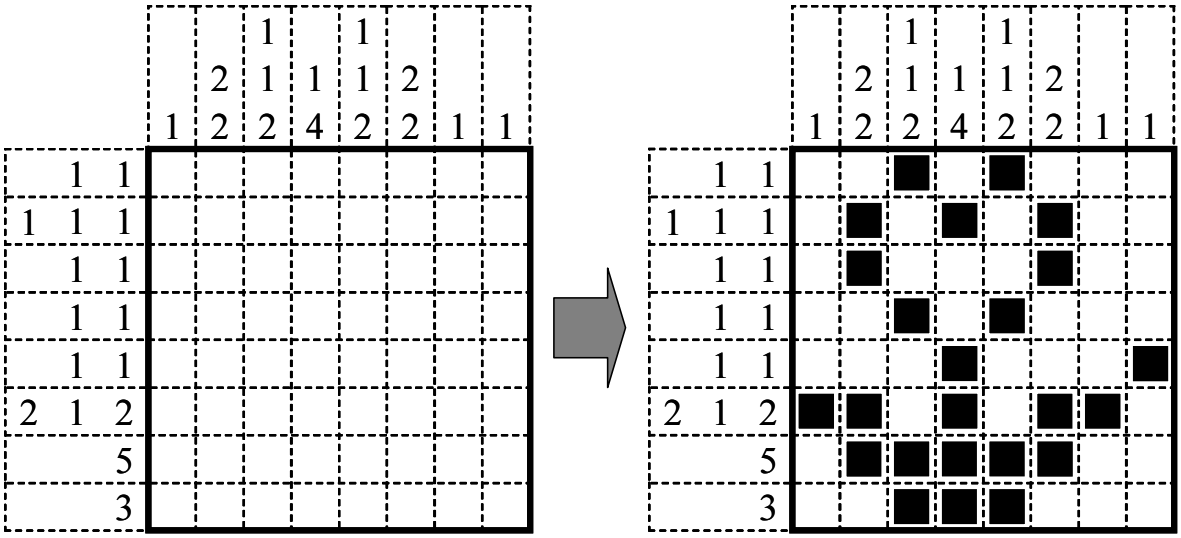


図 7.2　Example of constraint problem – Nonogram puzzle

- 並列化の理論 [60](小笠原 *et al.*)etc によって機能分割 (Partitioning) をサポートできる．
- 高位レベルから低位レベルまで ET ルールという単一の言語で記述でき，上位のレベルから等価にプログラム変換が可能である．
- 以上の性質によって宣言的仕様記述から自動的に最適なシステムを生成する capability がある．

7.2　ET の計算モデル

ET 言語は一階述語論理に基づく書き換え規則である．アイデアを記述する抽象度がある一方で正当性の保証，アルゴリズムの設計，低レベル言語への変換などプログラム生成の能力も持っている．この章では ET で制約充足問題を記述し解く方法について説明する．

7.2.1　制約充足問題パズル

例として「お絵かきロジック」と呼ばれるパズル[*1] を用いる (図 7.2)．このパズルはマトリクスを数列条件によって塗ってゆく制約充足問題である．

[*1] 英語では Nonogram や Griddler と呼ばれている．

ゲームのルールの説明

このマトリクスは，行ベクトルと列ベクトルからなる．各行の左側と各列の上側には，そのベクトルを塗りつぶすときに満足すべき条件が数列で表示されている．例えば 図 7.2 の上から 6 段目の行の数列 [2 1 2] の各数字は，左から右へ向かって連続して塗りつぶすマスの数を表しており，なおかつ各数字の間が一つ以上の空白で区切られなければならないことを表している．すなわち制約条件 [2 1 2] を満たす塗り方は，左から任意の場所のマスを**２個連続**して塗りつぶし (■■)，そこから右へ少なくとも 1 マス以上離れた場所のマスを**１個だけ**塗りつぶし (■)，そこから右へ少なくとも 1 マス以上離れた場所のマスを**２個連続**して塗りつぶす (■■) ことである．

数列 [2 1 2] が表す制約

$$[2\,1\,2] \;=\; \{\cdots \underset{2}{\blacksquare\blacksquare} \cdots \square \cdots \underset{1}{\blacksquare} \cdots \square \cdots \underset{2}{\blacksquare\blacksquare} \cdots\}$$

“· · ·” の部分には 0 個以上の連続した白いマス (□) が入る．
なおかつ，黒いマス (■) と白いマス (□) の個数の合計は
マトリクスの幅 (= 8) に等しい

ゲームを解く ET ルール

これを解く方法は下記の通りである:

1. 最初に，どこかの行ベクトルまたは列ベクトルを選択する．
2. 選択したベクトルの中に，制約条件から決定可能なセルがあれば決定する．
3. 上記を何度か繰り返すと，全てのセルが白または黒に決定される．

このパズルの質問を表す ans 節は図 7.3 で与えられる．pattern アトムの第一引数は数列条件を表しており，第二引数はベクトルの状態を表している．このアトムはベクトルが数列条件を満たすときに成功する．質問節のボディにある 16 個の pattern アトムは互いに変数を共有しており，全てのアトムが無矛盾となるように変数 V_{ij} の値を決定する．

図 7.4 はこのロジックパズルの問題を解く ET プログラムを抜粋したものである．*Seq* は制約条件の列をあらわしており，*Vec* はベクトルの状態をあらわす．関数 groundList は *Vec* がグラウンドかどうかを判定するビルトインである．手続き callSubGoal は特殊なビルトインであり，その引数にあるアトム fixpixel を実行する．手続き eq は第一引数と第二引数の単一化を試みるビルトインである．fixpixel アトムはユーザー定義アトムで，*Seq* 条件を満たすベクトルの中で *Vec* とマッチング可能なものの共通部分を求めることを意図する．

ET による計算プロセスについて説明する．与えられた節 Fig.7.3 のボディアトムの中でルールが適用可能なものを選択する．ET では適用可能なアトムが複数ある場合は非決定的に選択される．また適用可能なルールが複数ある場合にも非決定的に選択される．今仮

$$
\begin{aligned}
&\mathsf{ans}(V_{11}, V_{12}, \cdots, V_{88}) \leftarrow \\
&\quad /*\ \textit{horizontal vectors}\ */ \\
&\quad \mathsf{pattern}([\ \ 1,1], [V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}]), \\
&\quad \mathsf{pattern}([1,1,1], [V_{21}, V_{22}, V_{23}, V_{24}, V_{25}, V_{26}, V_{27}, V_{28}]), \\
&\quad \mathsf{pattern}([\ \ 1,1], [V_{31}, V_{32}, V_{33}, V_{34}, V_{35}, V_{36}, V_{37}, V_{38}]), \\
&\quad \mathsf{pattern}([\ \ 1,1], [V_{41}, V_{42}, V_{43}, V_{44}, V_{45}, V_{46}, V_{47}, V_{48}]), \\
&\quad \mathsf{pattern}([\ \ 1,1], [V_{51}, V_{52}, V_{53}, V_{54}, V_{55}, V_{56}, V_{57}, V_{58}]), \\
&\quad \mathsf{pattern}([2,1,2], [V_{61}, V_{62}, V_{63}, V_{64}, V_{65}, V_{66}, V_{67}, V_{68}]), \\
&\quad \mathsf{pattern}([\ \ 5], [V_{71}, V_{72}, V_{73}, V_{74}, V_{75}, V_{76}, V_{77}, V_{78}]), \\
&\quad \mathsf{pattern}([\ \ 3], [V_{81}, V_{82}, V_{83}, V_{84}, V_{85}, V_{86}, V_{87}, V_{88}]), \\
&\quad /*\ \textit{vertical vectors}\ */ \\
&\quad \mathsf{pattern}([\ \ 1], [V_{11}, V_{21}, V_{31}, V_{41}, V_{51}, V_{61}, V_{71}, V_{81}]), \\
&\quad \mathsf{pattern}([\ 2,2], [V_{12}, V_{22}, V_{32}, V_{42}, V_{52}, V_{62}, V_{72}, V_{82}]), \\
&\quad \mathsf{pattern}([1,1,2], [V_{13}, V_{23}, V_{33}, V_{43}, V_{53}, V_{63}, V_{73}, V_{83}]), \\
&\quad \mathsf{pattern}([\ 1,4], [V_{14}, V_{24}, V_{34}, V_{44}, V_{54}, V_{64}, V_{74}, V_{84}]), \\
&\quad \mathsf{pattern}([1,1,2], [V_{15}, V_{25}, V_{35}, V_{45}, V_{55}, V_{65}, V_{75}, V_{85}]), \\
&\quad \mathsf{pattern}([\ 2,2], [V_{16}, V_{26}, V_{36}, V_{46}, V_{56}, V_{66}, V_{76}, V_{86}]), \\
&\quad \mathsf{pattern}([\ \ 1], [V_{17}, V_{27}, V_{37}, V_{47}, V_{57}, V_{67}, V_{77}, V_{87}]), \\
&\quad \mathsf{pattern}([\ \ 1], [V_{18}, V_{28}, V_{38}, V_{48}, V_{58}, V_{68}, V_{78}, V_{88}]).
\end{aligned}
$$

図 7.3　Query clause to solve the puzzle of Fig.7.2

に節の中の下記のアトムに着目してみる．

$$\mathsf{pattern}(\,[2,1,2],\ [V_{61}, V_{62}, V_{63}, V_{64}, V_{65}, V_{66}, V_{67}, V_{68}]\,)$$

このアトムの第二引数は groundList ではないため $\boldsymbol{R}_{1b}$ が適用可能である．このアトムにこのルールが適用されるとき実行部で callSubGoal が実行される．callSubGoal は高階ビルトイン手続きで fixpixel を実行する．fixpixel により Vec の中で確定できる個所が発見される．今の例では変数 V_{62} と V_{67} が 1 に確定できる[*2] ので，Com には下記が返される．

$$Com = [\ \_, 1\ , \_, \_, \_, \_, 1\ , \_\ ]$$

fixpixel が成功すると callSubGoal が成功し eq が実行される．その結果 Vec の中の V_{62} と V_{67} が 1 に特殊化される．実行部が全て成功すると pattern アトムが置き換えられ query

[*2] なぜなら Vec とマッチング可能で数列条件 $[2,1,2]$ を満たす全てのベクトル “■■□■□■■□”, “■■□■□□■■”, “■■□□■□■■”, “□■■□■□■■” は共通して V_{62} と V_{67} の場所が 1 である．

R_{1a}: pattern(Seq, Vec), { groundList(Vec) }
==> .

R_{1b}: pattern(Seq, Vec), { not(groundList(Vec)) }
==> { callSubGoal(fixpixel(Seq, Vec, Com)), eq(Com, Vec) },
pattern(Seq, Vec) .

R_2: fixpixel(Seq, Vec, Com)
==> partialsearch(0, 63, $Seq, Vec, Pat1$),
partialsearch(64, 127, $Seq, Vec, Pat2$),
partialsearch(128, 191, $Seq, Vec, Pat3$),
partialsearch(192, 255, $Seq, Vec, Pat4$),
appendall($[Pat1, Pat2, Pat3, Pat4]$, Pat),
pat_intersection(Pat, Com) .

R_{3a}: appendall($[\,], Z$) ==> { $Z = [\,]$ } .
R_{3b}: appendall($[A|X], Z$) ==> appendall($X, Z1$), append($A, Z1, Z$) .

R_4: partialsearch(N_1, N_2, Seq, Vec, Pat)
==> $\cdots$ ***detail is omitted*** $\cdots$
このアトムは N_1 から N_2 の範囲のベクトルの
中で Seq と Vec の条件を満たすものを全て探し
それらを要素とするリスト Pat を返す.

R_5: pat_intersection(Pat, Com)
==> $\cdots$ ***detail is omitted*** $\cdots$
このアトムは Pat の要素の共通パターンを計算
する.

図 7.4 ET program to solve the puzzle of Fig.7.2 (snippet)

節は下記のように書き換えられる．

$$\begin{array}{l}\mathsf{ans}(V_{11},\cdots,V_{62},\cdots,V_{67},\cdots,V_{88})\\ \quad\leftarrow\quad \vdots\ (\textit{omitted})\\ \qquad \mathsf{pattern}([2,1,2],[V_{61},V_{62},V_{63},V_{64},V_{65},V_{66},V_{67},V_{68}]),\\ \qquad\qquad \vdots\ (\textit{omitted})\end{array}$$

$$\Downarrow \textbf{rewritten by } \boldsymbol{R}_{1b}$$

$$\begin{array}{l}\mathsf{ans}(V_{11},\cdots,\mathbf{1},\cdots,\mathbf{1},\cdots,V_{88})\\ \quad\leftarrow\quad \vdots\ (\textit{omitted})\\ \qquad \mathsf{pattern}([2,1,2],[V_{61},\mathbf{1},V_{63},V_{64},V_{65},V_{66},\mathbf{1},V_{68}]),\\ \qquad\qquad \vdots\ (\textit{omitted})\end{array}$$

以下同様に次々とルールによって節が書き換えられてゆく．第二引数が ground の pattern アトムは $\boldsymbol{R}_{1a}$ によって空に置き換えられるので節のボディアトムは次第に減ってゆく．そして最終的にボディアトムが全て空になって単位節が得られると計算は終了する．このとき全ての V_{ij} の値は確定する．

高階ビルトイン callSubGoal が引数の fixpixel を計算する方法も同様に行われる．このビルトインは，まず下記の子供の節を生成する．

$$\begin{array}{l}ans([2,1,2],[V_{61},V_{62},V_{63},V_{64},V_{65},V_{66},V_{67},V_{68}],Com)\\ \quad\leftarrow \mathsf{fixpixel}([2,1,2],[V_{61},V_{62},V_{63},V_{64},V_{65},V_{66},V_{67},V_{68}],Com)\ .\end{array}$$

そして，この生成された子供の節は親の節とは別のワールドで実行が開始される．最終的に単位節が得られると callSubGoal は成功する．生成された子供の節と親の節は変数を共有しているので，子供の計算で変数への代入がおこなわれると親の節の変数に反映される．

7.3 分散協調システムと ET

この章では前章の ET の計算方法を参照しながら，ET の計算モデルと分散協調システムの親和性について議論する．

7.3.1 多様な並列化

計算を効率よく計算するためには並列化が重要である．しかも与えられる分散協調システムによって効率的な並列化は異なるので様々な並列化を選択できなければならない．

ET の計算モデルでは下記に示す二つの並列化の組み合わせによって実行を容易に様々な粒度に分割して並列計算できる．それを例題を用いて説明する．

まず一つ目の並列化は**アトム並列化**である．この並列化は同時に複数のアトムを評価

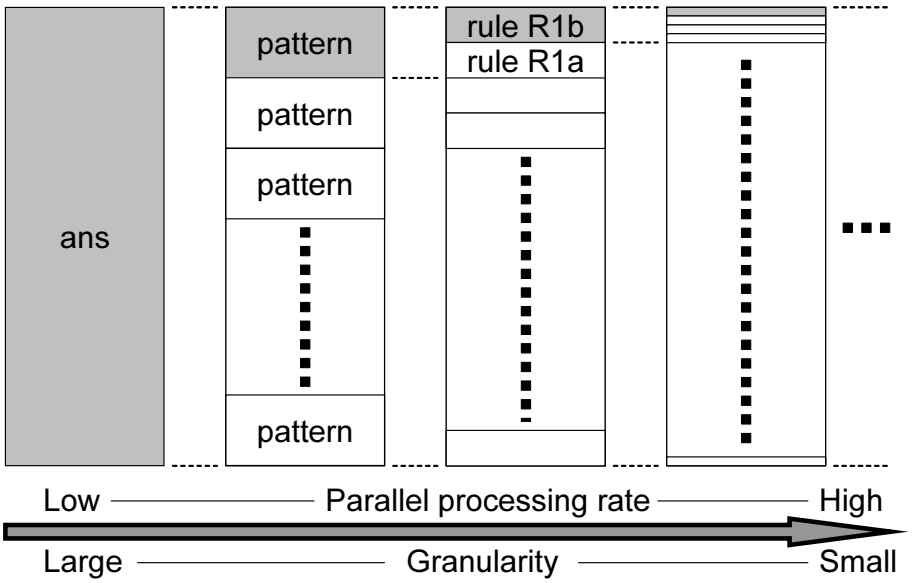


図 7.5　Granularity of parallelism

する．例えば図 7.3 の query のボディにある pattern アトムを並列に評価しても計算結果は矛盾しない．同様に fixpixel アトムを計算する $\boldsymbol{R}_2$ のボディには互いに独立な 4 つの partialsearch アトムがあり並列に評価できる．

二つ目の並列化は**ルール並列化**である．これは複数のルールの条件部を並列に計算する並列化である．例題では pattern アトムを計算するルールは $\boldsymbol{R}_{1a}$ と $\boldsymbol{R}_{1b}$ の二つある．どちらのルールが適用できるかどうかはルールの条件部を実行してみなければ分からない．これらの条件部を順番に試すより並列に計算してしまえばルール適用が高速化できる．また実行部も条件部と並列に計算しておけば更に高速化できる．一般にこの並列化は適用されないルールも計算するので投機的 (speculative) である．ルールの並列適用が可能なケースでは投機的にならないこともある．

このように ET の実行はアトム並列化とルール並列化のコンビネーションによって様々な粒度の並列実行に分割できる．この様子を図に表したものが図 7.5 である．この豊富な分割の仕方の中から与えられたシステムに最適な選択すればよい．

7.3.2　多様な機能分割

クライアントの処理をサーバーへ分散させたり，ハードウエアを使って計算をアクセラレートするには適切に機能を分割できなければならない．特に回路化するには FPGA の容量や記述が回路化可能であることなど制約が厳しい．与えられる環境に合わせて様々な分割点が選択できる必要がある．

ET は多様な並列化が可能なのだけでなく，処理を様々なレベルに分割できる．それゆえ，その中から与えられた分散協調計算システムにマッピング可能な最適な機能分割を

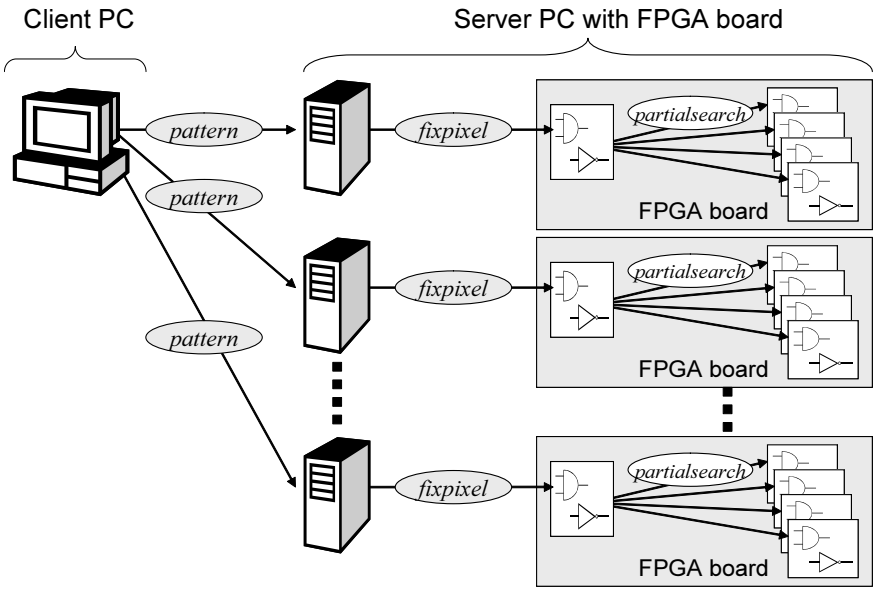


図 7.6　Mapping ET computation to parallel systems

見つけることが可能である．例えば図 7.6 は図 7.1 のトポロジーにマッピングした例である．例ではクライアントからサーバーへ依頼する処理の分割点は pattern アトムである．このアトムを複数のサーバーへ並列に依頼することでクライアントの負荷を減らす．サーバーから回路へ処理を依頼する分割点は回路化可能な記述の中から FPGA の容量と通信のオーバーヘッドを考慮して決める．この例ではサーバーから回路へ処理を依頼する分割点を fixpixel アトムとし，回路の並列計算性を活かして partialsearch を並列に計算している．

7.3.3　発展性

ET の記述は様々な変換やプログラム生成が可能である．現在進行中の研究としては背景知識 ID からルールを自動生成する研究や，ET ルールによってプログラムのアルゴリズムを設計する研究などがある．先行研究の回路生成の研究 [53] は，協調システムを自動生成するのに利用できる．ET は宣言的仕様から様々な可能性のシステムを生成できる理論でもある．システム生成問題やシステム最適化問題は ET 空間での探索問題に帰着される．

このように ET を用いて問題を記述すると，分散協調システムに必要なプログラムや回路を自動生成できる可能性を持っている．分散並列協調計算システムに必要なソフトウエアと回路を生成する枠組みが作れる (図 7.7).

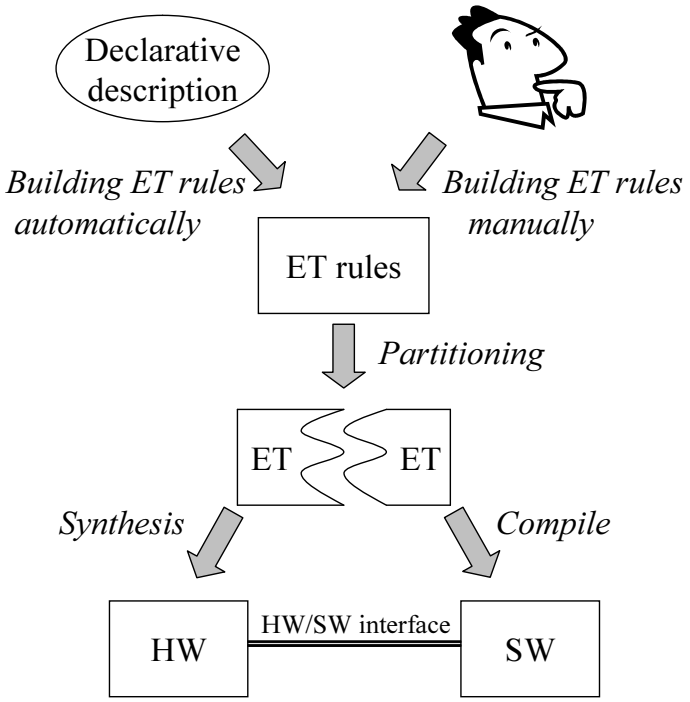


図 7.7　Co–design framework using ET

7.4　協調計算サーバーシステム

前章では制約充足問題を分散協調システムで解く場合に，ET によって記述することの利点を述べた．しかし実際に図 7.6 のトポロジーを容易に実現するためには ET から様々なハードウエアを呼び出す仕組みが必要である．このためのサーバーシステムを製作した．

7.4.1　構造

この柔軟なシステムは大きく分けて３つのパートから成っている．

■PCI Board with FPGA　一つは *HwModule Board* [58, 59] と呼ばれる回路ボードである．*HwModule Board* は動的に書き換え可能なユーザー回路と，回路とソフトウエアとの通信機能を提供する．このボード上には 1 メガゲートの FPGA である *Xilinx Spartan*^{TM}*–3* が４つ搭載されている．このうちのひとつは PCI バスコントローラやユーザー回路と PC の通信を管理する *System FPGA* である．残りの３つはユーザーが自由に使うことができる *User FPGA* である．HwModule Board とソフトウエアとの通信手段は付属の管理プログラム *ObjectManager* とデバイスドライバによって提供されている．

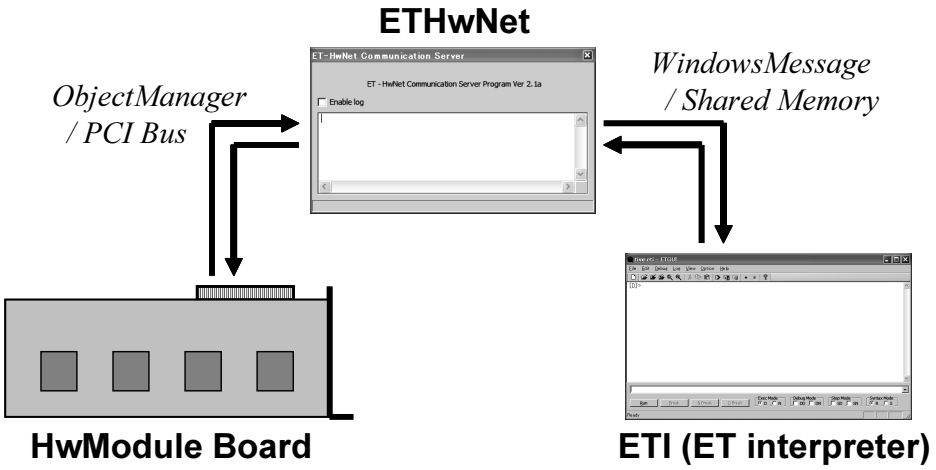


図 7.8　Architecture of the cooperative system

■Server Program　*ETHwNetServer* は，HwModule Board を利用するために本研究の中で開発されたサーバープログラムである．ソフトウエアからユーザー FPGA 中の回路を自由に再構成したり，ソフトウエアと回路の通信を仲介するための汎用のインターフェースを提供する．サーバープログラム *ETHwNetServer* のおかげで，ET プログラムは単なるアトム呼び出しによってユーザー回路を利用することができる．

■ET Built-in Atoms　ET プログラムは ET インタプリタ *ETI*[18–20] 上で動作する．ET プログラムが *ETHwNetServer* を介して回路を利用するため，ET インタプリタ (ETI) に built-in atom が追加されている．ET プログラムの中でこの built-in atoms が評価されるとき ETI からサーバーへリクエストが送られる．サーバーが ETI に結果を返すと，この built-in atoms の引き数にある変数へ結果が代入される仕組みである．この追加されたビルトインはサーバーと通信するために *WindowsMessage* と *SharedMemory* を使っている．

7.4.2　例題１ – Puzzle (size=8)

このサーバーシステムの性能をテストするため 図 7.2 (104 ページ) の問題を用いて実験した．テストはサーバー PC とハードウエアの協調計算の性能を測ることを目的としているのでクライアント PC は使わず 1 台のサーバー PC でテストした．

協調計算の効果をテストするため，2 種類の異なる環境でテストした．一つはソフトウェアのみで実行した場合であり，もう一つはデジタル回路とソフトウェアの協調計算を行った場合である．デジタル回路は並列計算の能力を活かし partialsearch アトムを 64 個並列に実行している．下記に結果を示す．

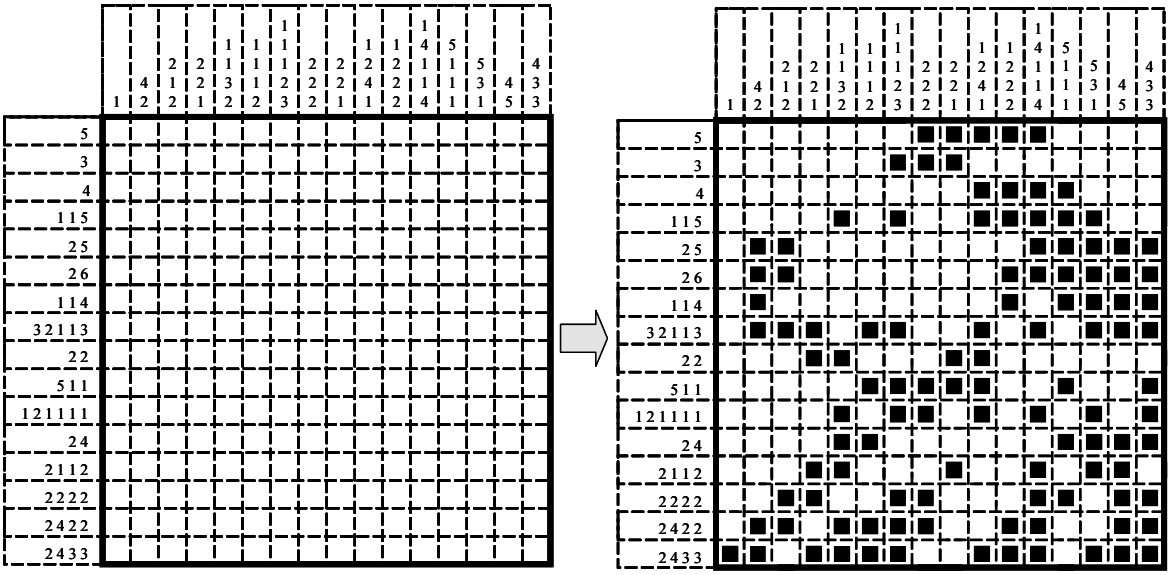


図 7.9　Logic Puzzle 16×16

configuration	time [sec]
SW only	0.110
SW + HW	0.031

実験環境は下記のとおりである．

PC : Pentium4 2.4GHz, 1GB memory
OS : Windows XP Professional SP2
HW : HwModule Board V2 with XC3S1000, 33MHz clock

ハードウエアを用いても，わずか 3.5 倍の差しか得られなかった．この例題ではソフトウエア単体で計算しても十分に短い時間で解けるので，全実行時間に占める通信時間の割合が大きかったと考えられる．

7.4.3　例題 2 – Puzzle (size=16)

続いて 図 7.9 に示す問題を解かせた．この問題は図 7.2 と比べて問題サイズはわずかに 2 倍だが，計算量が莫大でありソフトウエアだけで解くと非常に時間がかかる．この例題は協調計算システムの強みを活かせる良い例である．結果は以下のとおりである．

configuration	time [sec]
SW only	8,701.502
	(= 2h25m1.502s)
SW + HW	0.704

今回は計算量が莫大になったためにソフトウエア単体では計算時間が大きく，それゆえ通信時間のペナルティを払っても回路を使って計算することに十分なアドバンテージがあった．HwModule Board 上のデジタル回路の動作クロックはわずか 33MHz である．しかし上記の結果から，協調計算ではデジタル回路によって 12,360 倍のスピードアップが実現されていることが分かる．

7.5 この章のまとめ

制約充足問題を分散並列環境で解くために ET を用いた枠組みを提案した．この枠組みで，ET を用いると制約充足問題の記述が容易であること，並列計算が容易に切り出せること，機能分割するのは容易であることを示した．そしてこれらの性質によって分散並列協調システムの生成が可能なことを示した．また実際に実験システムを作り，ロジックパズルを用いてこのシステムのパフォーマンスを測定した．

第 8 章

結論

8.1 本研究の結論

提案フレームワークは仕様とターゲットの間のギャップを ET 空間で埋め、ET 空間内でプログラム変換という一貫した基本原理を用いて ET ルールを変換するという方針である。このフレームワークは回路生成だけでなく、他のターゲットへの変換にも利用できる普遍的なものになっている。

8.1.1 本手法の特色

本研究は，既存の手法や研究に対して下記に示す特色を持つことができた．

■HDL による回路設計　従来の VHDL[16], Verilog HDL[17] はモジュールの接続関係 (ネットリスト) やフリップフロップへの代入アクションを記述する．この手法による設計の難しさは，複数のモジュールが並列に動作しており，それらの動作を把握していなければ正しい動作を記述できないことである．

本研究は，確定節による宣言的仕様を記述するだけで回路が生成可能となる．宣言的仕様には答えが満たすべき充足関係が記述されていればよい．したがって「動作」を考える必要はない．そして ET ルールをマージする過程でデジタル回路の利点である並列計算を抽出することができる．

■述語論理による回路設計　Waterloo 大学の研究 (P.Gaboury [30]) は，論理プログラミングと同様に確定節による宣言的仕様を手続きと考えて回路を生成している．この手法ではアトムをひとつの回路モジュールと考え，アトムの引数を回路モジュールのインターフェースととらえている．そして論理変数は回路モジュールを接続する配線である．

本研究では，確定節を状態として捉えている．すると，アトムは単に記憶領域をグループ化したものであり，アトムの引数が記憶領域 (レジスタ，F/F) である．宣言的仕様に対して正しいならば，どのようなプログラム変換も可能なため，回路の実現方法の自由度を高くすることができる．

■TRS による回路設計　MIT による項書き換え系を用いた研究 (Hoe, *et al.*[31, 32] , Rosenband[33, 34]). また，筑波大学の研究 (Kobayashi, *et al.*[61]) もある．これらは，ガード条件付きの書き換え規則によって動作を記述しており，本研究が ET ルールという書き換え規則によって回路を記述しているのと似たアイデアとなっている．しかし，本研究は ET ルールを宣言的仕様から自動生成することを前提としており，更なる高位の設計手法となっている．

■システム記述言語による協調計算システムの設計　協調計算システムを記述する言語として SystemC[28], System Verilog[29] などがある．SystemC は C/C++ 言語の拡張であり，System Verilog は Verilog HDL の拡張である．

これらの言語では機能分割 (回路とソフトの役割を分けること) が難しいが，ET ルールは各ルールが独立であるため機能分割が容易になっている．

8.1.2　今後の課題と発展

本論文の手法は「仕様記述」と「生成された物」の間の正当性が数学的に保証される枠組みを重視している．このため，宣言的仕様と ET ルールの間の正当性が保証される「等価変換理論」の枠組みを用いた．ET ルールはインタラクティブな動作なども記述できる非常に記述力の高い言語である．しかし確定節による宣言的仕様が記述できるものは制約充足問題などの求解問題に限られる．このため ET ルールの記述力の高さを活かしきれていない．この部分に多くの可能性と今後の研究の余地が残されている．下記に今後の課題と発展を挙げる．

■回路生成手法の課題と発展　本書で紹介したルールの変換手法は，宣言的仕様からデジタル回路が生成可能であることをデモするためアドホックに開発されたものである．特にメタ計算手法は実験的であり理論的に整備されていない．ただし，本書では，このアドホックな手法を考察し議論することで，プログラム変換を代数系変換として捉えることを発見している．このことにより，プログラム変換の自由度が広がり様々な変換手法を開発する基礎を与えることができた．

協調計算システムの生成に関しては，SW と HW への機能分割の自動化を実現する課題がある．このためには，回路化可能なルールの発見または判定が必要になる．書き換え規則は各ルールが独立しているため，従来の HDL(VHDL,Verilog,etc.) に比べて機能分割が容易である．この特徴を活かして効率重視やコスト重視など目的に応じた柔軟な分割方法の実現も期待できる．

6 章で指摘したように，本研究で提案した ET ルール変換の枠組みでは用途に応じてポリシーの作り方やストラテジーの立て方にバリエーションが考えられる．これらのバリエーションを増やすことも今後の研究課題である．

■理論の課題と発展　本書で紹介したルール変換手法は一つの節 (節の単集合) を状態としている．このため，ET ルールのマルチボディを扱っていない．しかし，実際の ET ルー

ルの計算ではマルチボディルールによって計算の可能性が分岐し，一般に状態は複数の節からなる集合となる．もし，節の集合 (複数の節の塊) を一つの状態として考えることが可能ならば，この状態を代数系変換に適用することでプログラム変換が可能となる．

■仕様記述の課題と発展　本書の範囲は，宣言的仕様が定義する問題を解く回路/システムを生成する研究である．しかし，一般的なデジタル回路の応用範囲は広く，例えばタイムクリティカル (リアルタイム) な回路やインタラクティブな回路などもよく使われる．現在の宣言的仕様では，このような回路の仕様を記述できない．

本研究の動機には，仕様と生成物との間で正当性が数学的に保証される枠組みの構築がある．この枠組みの中で上記の動作をする回路を生成するための仕様記述方法の研究が必要である．

■モジュールおよびマクロ機能の利用の課題　本研究では，回路全体をフラットな階層として扱う．このため回路をモジュール化して再利用することや，FPGA などに内蔵されたマクロ機能 (埋め込み乗算器や埋め込み RAM など) を利用することができない．

モジュールを利用するための取り組みとして，モジュールの動作 (ビヘイビア) を ET ルールで記述することが考えられる．

■デジタル回路，CPU，ソフトウェアを含めた生成の可能性　昨今の FPGA の容量は大きく，簡単な CPU を HDL によって記述し合成しても容量が余るほどである．ET ルールによって CPU の動作を記述し、その CPU 上で動作するソフトウエアも含めて生成する研究が考えられる．CPU とプログラムを同時に生成するには，命令セットとプログラムの組み合わせ方も課題になる．

謝辞

北海道大学 大学院情報科学研究科 複合情報学専攻 複雑系工学講座 混沌系工学研究室 小野教授には，本研究を進めるにあたり，深い議論を通してご指導いただくとともに終始健康面を気遣って下さり，また励ましを賜りました．深く感謝の意を表します．

北海道大学 大学院情報科学研究科長 栗原教授 ならびに 北海道大学 情報基盤センター長 山本強教授には，進路等につきましてご助言とご支援を賜りました．深く御礼申し上げます．また，公私にわたりお世話になった北海道大学大学院情報科学研究科の皆様に感謝いたします．

日本 IBM 時代に私を支え続けてくださり，いつも温かく，時に厳しく指導してくださり，現在の私の研究者としての礎を築いてくださった 朝田さん，木村さん，軽部さん，そして科学的かつ論理的な思考のトレーニングをしてくださった 日本 IBM の素敵な同僚の皆様とパートナー会社の皆様に心よりの感謝とお礼を申し上げます．

最後に，お世話になった全ての皆様に深く感謝申し上げます．

参考文献

[1] Kiyoshi Akama. Semantic processing of natural language sentences by equivalent transformation. *Hokkaido University Information Engineering Technical Report, HIER–LI–9225*, 1992.

[2] K. Akama, Y. Nomura, and E. Miyamoto. Semantic interpretation of natural language descriptions by program transformation. *Computer Software(Tokyo)*, Vol. 12, No. 5, pp. 45–62, 1995.

[3] K Akama, T Shimizu, and E Miyamoto. Solving problems by equivalent transformation of declarative programs. *Journal of the Japanese Society for Artificial Intelligence*, Vol. 13, pp. 944–952, 1998.

[4] Kiyoshi Akama and Ekawit Nantajeewarawat. Formalization of the equivalent transformation computation model. *JACIII*, Vol. 10, No. 3, pp. 245–259, 2006.

[5] Kiyoshi Akama and Ekawit Nantajeewarawat. Computation models and correctness relations. *Technical report of IEICE. SS*, Vol. 103, No. 583, pp. 7–12, 20040116.

[6] Kiyoshi Akama, Ekawit Nantajeewarawat, and Hidekatsu Koike. A class of rewriting rules and reverse transformation for rule–based equivalent transformation. *Electronic Notes in Theoretical Computer Science*, Vol. 59, No. 4, 2001.

[7] Kiyoshi Akama, Ekawit Nantajeewarawat, and Hidekatsu Koike. Program generation in the equivalent transformation computation model using the squeeze method. *Perspectives of System Informatics, Lecture Notes in Computer Science*, Vol. 4378, pp. 41–54, 2006.

[8] Kiyoshi Akama, Ekawit Nantajeewarawat, and Hidekatsu Koike. Program synthesis based on the equivalent transformation computation model. In *LOPSTR*, pp. 278–279, 2002.

[9] Kiyoshi Akama, Hidekatsu Koike, and Eiichi Miyamoto. Generating equivalent transformation rules from specifications of problems. *Transactions of Information Processing Society of Japan*, Vol. 40, No. 4, p. 74, 19990515.

[10] Katsunori Miura, Kiyoshi Akama, and Hiroshi Mabuchi. Generating *speq* rules based on automatic proof of logical equivalence. *International Journal of Computer Science*, Vol. 3, No. 3, pp. 190–198, 2008.

[11] Katsunori Miura, Hiroshi Mabuchi, and Kiyoshi Akama. Creation of ET rules from

logical formulas representing equivalent relations. *International Journal of Innovative Computing, Information and Control*, Vol. 5, No. 2, pp. 263–277, February 2009.

[12] Yoshitaka Nishida, Kiyoshi Akama, and Hidekatu Koike. An experimental program-generation system based on meta–computation. *Technical report of IEICE. SS*, Vol. 107, No. 392, pp. 31–36, 2007-12-10.

[13] Juris Hartmanis. *Algebraic structure theory of sequential machines (Prentice–Hall international series in applied mathematics)*. Prentice–Hall, Inc., Upper Saddle River, NJ, USA, 1966.

[14] Edward F. Moore. Gedanken–experiments on sequential machines. *Automata Studies*, pp. 129–153, 1956.

[15] George H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, Vol. 34, pp. 1045–1079, September 1955.

[16] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076–2002.

[17] IEEE Standard for Verilog Hardware Description Language, IEEE Std 1364–2005.

[18] Hidekatsu Koike, Kiyoshi Akama, and Hiroshi Mabuchi. Equivalent transformation language interpreter ETI. In *2001 IEEE 5th International Conference on Intelligent Engineering Systems (INES'01)*, pp. 155–160, Sept. 2001.

[19] Hidekatsu Koike, Kiyoshi Akama, and Hiroshi Mabuchi. A programming language interpreter system based on equivalent transformation. In *2005 IEEE 9th International Conference on Intelligent Engineering Systems (INES'05)*, pp. 283–288, Sept. 2005.

[20] ET rule interpreter
http://assam.iic.hokudai.ac.jp/eti/.

[21] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers part I*, Vol. 72, No. 9, pp. 593–599, November 1953.

[22] W.V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, Vol. 62, No. 9, pp. 627–631, November 1955.

[23] E.J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, Vol. 35, No. 5, pp. 1417–1444, November 1956.

[24] S. J. Hong, R. G. Cain, and D. L. Ostapko. Mini: A heuristic approach for logic minimization. *IBM Journal of Research and Development*, Vol. 18, pp. 443–458, 1974.

[25] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984.

[26] C.S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, Vol. EC–13, No. 2, pp. 14–17, Feb. 1964.

[27] Andrew D. Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, Vol. 4, pp. 236–240, 1951.

[28] IEEE Standard SystemC Language Reference Manual, IEEE std 1666–2005.

[29] IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800–2005.

[30] Pierre Gaboury. *VLSI architecture design using predicate logic*. PhD thesis, University of Waterloo, Waterloo, Ont., Canada, Canada, 1990.

[31] James C. Hoe and Arvind. Hardware synthesis from term rewriting systems. In *VLSI '99: Proceedings of the IFIP TC10/WG10.5 Tenth International Conference on Very Large Scale Integration*, pp. 595–619, Deventer, The Netherlands, 2000. Kluwer, B.V.

[32] James C. Hoe and Arvind. Synthesis of operation–centric hardware descriptions. In *IEEE/ACM International Conference on Computer-aided design (ICCAD)*, pp. 511–518, 2000.

[33] Daniel L. Rosenband. The ephemeral history register: flexible scheduling for rule–based designs. In *MEMOCODE*, pp. 189–198. IEEE, 2004.

[34] Daniel L. Rosenband. Hardware synthesis from guarded atomic actions with performance specifications. In *Proceedings of the 2005 IEEE/ACM International conference on Computer–aided design*, ICCAD '05, pp. 784–791, Washington, DC, USA, 2005. IEEE Computer Society.

[35] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, UK, 1998.

[36] Elliott Mendelson. *Introduction to Mathematical Logic (Discrete Mathematics and Applications)*. Chapman & Hall, 5 edition, 8 2009.

[37] Dirk van Dalen. *Logic and Structure (Universitext)*. Springer, 4th ed. 2004. corr. 2nd printing edition, 3 2004.

[38] 戸田山和久. 論理学をつくる. 名古屋大学出版会, 2000.

[39] J. W. Lloyd. *Foundations of logic programming; 2nd Extended Edition*. Springer–Verlag New York, Inc., New York, NY, USA, 1987.

[40] 有川節夫, 原口誠. 述語論理と論理プログラミング (知識工学講座). オーム社, 1988.

[41] 森下真一. 知識と推論 (情報数学講座). 共立出版, 1994.

[42] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, Vol. 16, pp. 14–21, 1951.

[43] D. L. Bowen. *DECSystem–10 PROLOG User's Manual*. Dept. of Artificial Intelligence, Univ. of Edinburgh, 1983.

[44] Robert A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pp. 569–574, 1974.

[45] Krzysztof R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. ACM*, Vol. 29, pp. 841–862, July 1982.

[46] 赤間清, 川口雄一, 宮本衛市. 項領域における包含制約の等価変換. 人工知能学会誌, Vol. 13, No. 2, pp. 274–282, 1998-03-01.

[47] 赤間清, 繁田良則, 宮本衛市. 特殊化システムの拡張による知識表現系の変更. 人工知能学会誌, Vol. 13, No. 1, pp. 131–138, 1998.

[48] 吉田忠行, 赤間清, 宮本衛市. 一階論理制約による宣言的記述の拡張. 情報処理学会研究報告. ICS, [知能と複雑系], Vol. 98, No. 65, pp. 17–24, 1998.

[49] C. Anutariya, V. Wuwongse, K. Akama, and E. Nantajeewarawat. RDF declarative description (RDD): A language for metadata. *Journal of Digital Information*, Vol. 2, No. 2, 2001.

[50] E. Nantajeewarawat, V. Wuwongse, C. Anutariya, K. Akama, and S. Thiemjarus. Toward reasoning with unified modeling language diagrams based–on extensible markup language declarative description theory. *International Journal of Intelligent Systems*, Vol. 19, pp. 89–98, 2004.

[51] Vilas Wuwongse, Kiyoshi Akama, Chutiporn Anutariya, and Ekawit Nantajeewarawat. A data model for xml databases. *Journal of Intelligent Information Systems*, Vol. 20, pp. 63–80, January 2003.

[52] Hiroshi Mabuchi, Kiyoshi Akama, and Toshihiro Wakatsuki. Equivalent transformation rules as components of programs. *International Journal of Innovative Computing, Information and Control*, Vol. 3, No. 3, pp. 685–696, June 2007.

[53] Hiroshi Yoshikawa, Kiyoshi Akama, and Hiroshi Mabuchi. Logic circuit synthesis preserving correctness using ET rules. *WSEAS Transactions on Circuits and Systems*, Vol. 6, No. 5, pp. 465–472, May 2007.

[54] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, Vol. 2, No. 42, pp. 230–265, 1936.

[55] Yoshinori Shigeta, Kiyoshi Akama, Hiroshi Mabuchi, and Hidekatsu Koike. Converting constraint handling rules to equivalent transformation rules. *JACIII*, Vol. 10, No. 3, pp. 339–348, 2006.

[56] Hiroshi Yoshikawa, Kiyoshi Akama, Hiroshi Mabuchi, and Rika Satoh. Flexible hardware–software cooperation system with HwModule board and co-design framework by ET. In *Proceedings of the 7th WSEAS International Conference on Applied Computer and Applied Computational Science*, pp. 248–253, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[57] Hiroshi Yoshikawa, Kiyoshi Akama, and Hiroshi Mabuchi. ET–based distributed cooperative system. *International Journal of Innovative Computing, Information & Control (IJICIC)*, Vol. 5, No. 12, pp. 4655–4666, Dec 2009.

[58] Kenji Kudo, Yoshihiro Myokan, Winh Chan Than, Shinji Akimoto, Takashi Kanamaru, and Masatoshi Sekine. Hardware object model and its application to the image processing. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, Vol. 87, No. 3, pp. 547–558, 20040301.

[59] Yoshihiro Myokan, Kenji Kudo, W Chanthan, Masatsugu Fujita, and Masatoshi Sekine. Real–time image processing system using HwObject. *IEIC Technical Report*, Vol. 103, No. 736, pp. 25–30, 2004.

[60] H. Ogasawara, K. Akama, H. Koike, H. Mabuchi, and Y. Saito. Parallel processing method based on equivalent transformation. *Intelligent Engineering Systems, 2005. INES '05. Proceedings. 2005 IEEE International Conference on*, pp. 111–116, Sept.16–19 2005.

[61] Noriyuki Kobayashi, Atusi Maeda, and Yoshinori Yamaguchi. Prototype implementation of using term rewriting system to design hardware. *IEICE technical report. Computer systems*, Vol. 104, No. 240, pp. 55–60, 2004.