佳作論文

# ハードウェアの機能設計段階における検証†

丸　山　文　宏††

コンピュータの LSI 化に伴い，論理設計の高信頼化が一層強く要請されるようになってきた．我々は，この要請に答えるため，機能設計の段階にハードウェア記述用言語 DDL による正式な記述を導入し，シミュレータおよび DDL の機能的記述から回路設計のための情報を抽出・整理するトランスレータの開発を行ってきたが，ここでは，論理設計の検証についての研究を行った．この分野では，最近2,3の研究結果が報告されているが，いずれも定理の証明やプログラムのテスト方法の援用の域にある．

この研究では，ハードウェア設計の伝統的手段の一つである状態遷移表現を充分に活用した新しい検証法を提案する．本論文では，状態遷移表現を特徴とする DDL の機能的記述からトランスレータによって抽出・整理された情報を知識ベースとして，ハードウェアの機能記述に含まれる矛盾の発見や与えられた仕様を満たすことの検証を行う方法を述べる．本方式は，特に，大型コンピュータの論理設計の検証を目標にしており，設計ミスが多く問題となっているユニット間のインタフェースの検証に有効なアルゴリズムを与える．現在，会話形実験システムを用いてその評価を行っているが，シミュレーションでは検出しにくいタイプの設計ミスの発見や，設計者の予期しなかった状況が起った際の原因解明に有効性を発揮すると思われる．

## 1. まえがき

コンピュータの LSI 化が進み VLSI 化が目指されている今日，現場における技術的変更は困難になり，論理設計の高信頼化が一段と強く要請されるようになってきた．

従来，論理設計者は，仕様が決定すると，ブロック・ダイアグラム，ステート・ダイアグラム，タイム・チャート等を用いて機能設計を行い，これに基づいてゲートレベルの設計を行っていた．これらの方法は完全にシステムを表現するものではなく，設計者間のコミュニケーション等に問題があった．また，設計のチェックもゲートレベルのシミュレーションに待たなければならなかった．

このような事情を踏まえて，我々は，機能設計の段階にハードウェア記述用言語 DDL (Digital system Design Language)[1),2)] を導入した．この段階で充分に設計をチェックした後ゲートレベルの設計を行うことにより，設計ミスを早期に発見でき，論理設計の信頼性が高まる．DDL の支援ソフトウェアとして，動作確認のためのシミュレータ，機能設計をゲートレベルに自動変換するトランスレータを開発した[3)]．

トランスレータによりゲートレベルの信頼性が高まり，論理設計の信頼性は，機能レベルの信頼性にかかって来ているが，シミュレーションですべての場合を調べることは実際上不可能である．また，大規模システムで問題となる，ユニット間のインタフェースに関するチェックなどをシミュレーションで行うことは難しい．これらの問題点を解決するためには，機能設計段階において，通常のシミュレーションとは異なる手法で検証することが必要である．

ハードウェアの検証に関しては，最近2,3の研究結果[4),5)]が報告されているが，いずれも定理の証明やプログラムのテスト方法の援用の域にある．しかし，プログラムが手続的であるのに対して，ハードウェアは非手続的に記述するのが自然であり，プログラム検証法の援用にも限界がある．

我々は，ハードウェア設計の伝統的手段の一つである状態遷移表現を充分に活用した新しい検証法を提案する．本論文では，前述のトランスレータにより抽出・整理された情報を知識ベースとして，ハードウェアの機能記述に含まれる矛盾の発見や与えられた仕様を満たすことの検証を行う方法を述べる．

本検証方式は，当初から大型コンピュータの論理設計の検証を目標としており，実際面からの検討を行っている．特に，大規模システムで設計ミスが多く問題となっているユニット間のインタフェースの検証等について考察している．

以下では，第2章で，機能設計に含まれる矛盾の例を挙げ，その検証（およびそれ以降の検証）を論理式が恒等的に偽であること，

$$(\text{論理式}) \equiv \phi, \tag{1.1}$$

の検証に帰着させるという本論文の統一的な方針を述

† Hardware Verification at Functional Design Stage by FUMIHIRO MARUYAMA (Information Processing Laboratory, Fujitsu Laboratories Ltd.).

†† (株)富士通研究所電子研究部門・情報処理研究部

べる．第3章では，(1.1) の検証のために時間を遡るアルゴリズムを提示し，第4章で，このアルゴリズムによって，与えられた仕様を満たしていることの検証が行われることを示す．第5章では実験システムについて簡単に報告する．

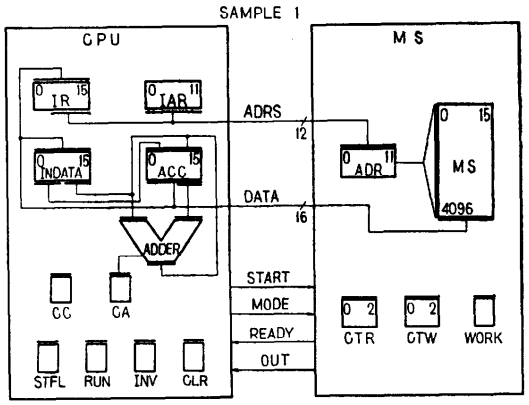



図 1 ブロックダイアグラム
Fig. 1 Block diagram (SAMPLE 1).

```
<SISTEM> SAMPLE1:
  <TIME> CLK(10).  /* CLOCK */
    <TERMINAL> START,MODE,ADRS(12),READY,OUT,DATA(16),
    <AUTOMATON> MS: CLK; /* AUTOMATON MS IS CONTROLLED BY CLOCK CLK */
      <STORAGE>  MS(4096,16).  /* 16 BITS 4096 WORDS */
      <REGISTER> ADR(12),WORK,CTR(3),CTW(3).
      <STATES>
        IDLE:       /* STATE IDLE */
              | START | ADR←ADRS,WORK←1,→RCI
              ;       →IDLE..
        RCI:        /* STATE RCI */
              CTR←CTR+1.
              | CTR:=1 | →READ
              ;        →RCI..
        READ:       /* STATE READ */
              OUT←1,CTR←0.
              | ¬MODE |  DATA←MS(ADR),WORK←0,→IDLE
              ;        →WRITE..
        WRITE:      /* STATE WRITE */
              MS(ADR)←DATA,→WCI.
        WCI:        /* STATE WCI */
              CTW←CTW+1.
              | CTW:=1 | WORK←0,CTW←0,→IDLE
              ;        →WCI..
      <END>.
    <END> MS.
    <AUTOMATON> CPU: CLK; /* AUTOMATON CPU IS CONTROLLED BY CLOCK CLK */
      <REGISTER> IR(16),IAR(12),INDATA(16),ACC(16),
                 CA,CC(2),RUN,CLR,INV,STFL.
      <STATES>
        IF0:        /* STATE IF0 */
              | RUN | START←1,ADRS←IAR,→IF1
              ;       →IF0..
        IF1:        /* STATE IF1 */
              | OUT | IR←DATA,IAR←IAR+1,→IF2
              ;       →IF1..
        IF2:        /* STATE IF2 */
              | IR(0:3):=1 | START←1,ADRS←ADR,→LOAD0..
              | IR(0:3):=2 | START←1,ADRS←ADR,STFL←1,→STORE0..
                      .
                      .
                      .
        STORE0:     /* STATE STORE0 */
              | OUT | →STORE1
              ;       →STORE0..
        STORE1:     /* STATE STORE1 */
              DATA←ACC,→STORE2.
        STORE2:     /* STATE STORE2 */
              | READY | STFL←0,→IF0
              ;         →STORE2..
                      .
                      .
                      .
    <END> CPU.
<END> SAMPLE1.
```

（オートマトン CPU の内部の詳しい記述は省略してある）

図 2 DDL による機能設計
Fig. 2 Functional design in DDL.

表 1 DDL の記号
Table 1 Symbols in DDL.

| 記号 | 意 味 |
|---|---|
| ¬ | 否定 (negation) |
| ∧ | 論理積 (logical product) |
| ∨ | 論理和 (logical sum) |
| = | 等号 (equality) |
| = | 結合 (connection) |
| ← | 転送 (transfer) |
| → | 転送 (transition) |
| $ | オートマトン名の修飾 |

## 2. 機能設計の記述とその中に含まれる矛盾

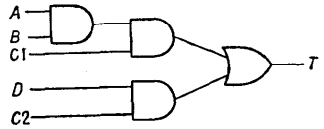
DDL は，ハードウェアの機能設計を記述するためのレジスタ・トランスファ・レベルの言語である．DDL で記述された機能設計の中には，矛盾と考えられるものが含まれることがある．ここでは，このような矛盾がないことを検証する問題を取り上げる．

### 2.1 D D L

最初に，状態遷移表現を特徴とするハードウェア記述用言語 DDL について，以後の議論に必要な範囲に限って説明する．図 1，図 2 に簡単な計算機の例 (SAMPLE 1) のブロックダイアグラム，DDL による機能設計を示す．表 1 には DDL で使われる記号を示す．

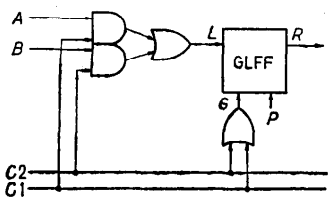
信号線あるいは端子をターミナル (TERMINAL) と呼ぶ．ターミナルに情報源(ソースと呼ぶ)から情報を出すオペレーションが結合 (CONNECTION) である．結合は，時間遅れなしに入力信号が出力に現れる理想的な組合わせ回路に対応する (図 3)．ただし，$|C1|T=A\wedge B.$ は，$C1$ が1ならばターミナル T に $A\wedge B$ を結合するという意味である．結合する値を指定されていない時のターミナルの値は0であると約束する．

フリップフロップのような記憶機能を持つ素子またはその集合体がレジスタ (REGISTER)，ストレジ




| C1 | $T=A\wedge B.$
| C2 | $T=D.$

図 3 ターミナルと結合
Fig. 3 Terminal and connection operation.




| C1 | R←A.
| C2 | R←B.

図 4 レジスタと転送
Fig. 4 Register and transfer operation.

(STORAGE) である．レジスタ (ストレジ) に情報源 (ソース) から情報をロードするオペレーションが転送 (TRANSFER) である．転送はあるクロック周期 (クロック周期を離散的な時刻と考えることができる) における入力が次のクロック周期 (次の時刻) の出力を決定する順序回路の動作に対応する (図 4)．ただし，図のレジスタ GLFF において，$L$は入力データを，$G$は入力条件を，$P$はクロックを表わしている．

ハードウェアの制御機能上一つのまとまりを持つ部分をオートマトン (AUTOMATON) と呼ぶ．オートマトンは常にそのステート (STATE) のうちのどれか1つのステートを取り，各ステートに対して記述されたオペレーションが実行される．次のクロック周期のステートを指定するオペレーションが転移(TRANSITION) である．

オペレーションに対する局部的な条件は IF 文で与えることができる：

|Be| $OP_t$.

または

|Be| $OP_t$; $OP_f$.

論理式 Be の値が 1 (真) の時オペレーション群 $OP_t$ が実行され，Be の値が 0 (偽) ならば $OP_f$ が実行される．

ここで図2の DDL の記述について簡単に触れておく．計算機 SAMPLE 1 は2つのオートマトン MS, CPU から構成され，クロック CLK により制御されている．ターミナル，ストレジ，レジスタの宣言の後，オートマトン MS のステートとそのステートで行われるオペレーションが記述されている．8行目から10行目にはステート IDLE で行われるオペレーションが示されている．そのオペレーションとは，ターミナル START の値が1なら，レジスタ ADR にターミナル ADRS の値を，レジスタ WORK に1を転送してステート RCY に転移し，START の値が0ならステート IDLE に留まるというものである．

回路の振舞が表現された DDL の記述を，ゲートレベルの設計のために，回路の構造的な情報に変える変換をトランスレーションという．トランスレーションによって，各ターミナル，レジスタ，ステートごとに，それに対するすべてのオペレーションがその行われる条件とともにまとめられ，編集される．ゲートレベルの設計のために考えられたトランスレーションがこれから述べる検証にも重要な働きをし，検証の際には (DDL の元の記述ではなく) トランスレーションリストが参照される．図5に図2に示した SAMPLE 1 をトランスレーションした結果 (の一部) を示す．レジスタ CTR に値を転送するオペレーションは図2にアンダーラインを引いて示した2つのものである．12行目のオペレーション (ステート RCY で行われる)，16行目のオペレーション (ステート READ で行われる) がトランスレーションによってまとめられ編集されて図5には No. 8-1, 8-2 に示されている．図5で CONDITION の欄が空欄のオペレーションは無条件に行われるオペレーションである．

### 2.2 機能設計に含まれる矛盾

DDL による機能設計に含まれる矛盾には次のようなものが考えられる．

(i) 1つのターミナル (またはレジスタ) に異なるソースを結合 (転送) する条件が排反でない．

(ii) 1つのステートから異なるステートに転移する条件が排反でない．

(iii) 1つのステートからのすべての転移条件の論理和が1にならない．

(i)の場合には，ターミナル (レジスタ) に同時に二つ以上の値が結合 (転送) されることがあり得る．(ii), (iii) の場合には，次の時刻のステートが2つ以上になったり決まらなかったりする．このような矛盾がゲートレベルではもはや矛盾として検出されなくなってしまうことがある．例えば，図3の機能設計 (DDL) は $C1$ と $C2$ が排反でない時(i)の矛盾を含むが，回路図については矛盾は検出されない．なぜなら，矛盾のない機能設計

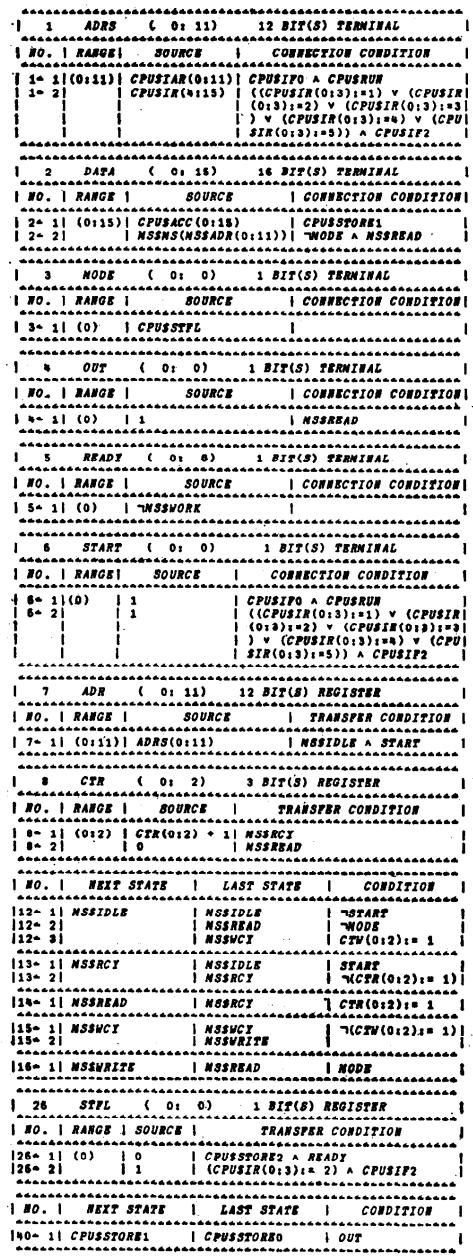
$|C1\wedge\neg C2|\quad T=A\wedge B.$

$|\neg C1\wedge C2|\quad T=D.$

$|C1\wedge C2|\quad T=(A\wedge B)\vee D.$

の回路図でもあるから，このことが，機能設計のレベルで検証を行う理由の1つにもなっている．

計算機 SAMPLE 1 の機能設計に矛盾がないことを



| 1 ADRS ( 0: 11) 12 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 1~ 1 | (0:11) | CPUSIAR(0:11) | CPUSIF0 ∧ CPUSRUN |
| 1~ 2 | | CPUSIR(0:15) | ((CPUSIR(0:3):=1) ∨ (CPUSIR(0:3):=2) ∨ (CPUSIR(0:3):=3) ) ∨ (CPUSIR(0:3):=4) ∨ (CPUSIR(0:3):=5)) ∧ CPUSIF2 |

| 2 DATA ( 0: 15) 16 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 2~ 1 | (0:15) | CPUSACC(0:15) | CPUSSTORE1 |
| 2~ 2 | | MSSNS(MSSADR(0:11)) | ¬MODE ∧ MSSREAD |

| 3 MODE ( 0: 0) 1 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 3~ 1 | (0) | CPUSSTFL | |

| 4 OUT ( 0: 0) 1 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 4~ 1 | (0) | 1 | MSSREAD |

| 5 READY ( 0: 0) 1 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 5~ 1 | (0) | ¬MSSWORK | |

| 6 START ( 0: 0) 1 BIT(S) TERMINAL | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | CONNECTION CONDITION |
| 6~ 1 | (0) | 1 | CPUSIF0 ∧ CPUSRUN |
| 6~ 2 | | 1 | ((CPUSIR(0:3):=1) ∨ (CPUSIR(0:3):=2) ∨ (CPUSIR(0:3):=3) ) ∨ (CPUSIR(0:3):=4) ∨ (CPUSIR(0:3):=5)) ∧ CPUSIF2 |

| 7 ADR ( 0: 11) 12 BIT(S) REGISTER | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | TRANSFER CONDITION |
| 7~ 1 | (0:11) | ADRS(0:11) | MSSIDLE ∧ START |

| 8 CTR ( 0: 2) 3 BIT(S) REGISTER | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | TRANSFER CONDITION |
| 8~ 1 | (0:2) | CTR(0:2) + 1 | MSSRCY |
| 8~ 2 | | 0 | MSSREAD |

| NO. | NEXT STATE | LAST STATE | CONDITION |
|---|---|---|---|
| 12~ 1 | MSSIDLE | MSSIDLE | ¬START |
| 12~ 2 | | MSSREAD | ¬MODE |
| 12~ 3 | | MSSWCY | CTR(0:2):= 1 |
| 13~ 1 | MSSRCY | MSSIDLE | START |
| 13~ 2 | | MSSRCY | ¬(CTR(0:2):= 1) |
| 14~ 1 | MSSREAD | MSSRCY | CTR(0:2):= 1 |
| 15~ 1 | MSSWCY | MSSWCY | ¬(CTR(0:2):= 1) |
| 15~ 2 | | MSSWRITE | |
| 16~ 1 | MSSWRITE | MSSREAD | MODE |

| 26 STFL ( 0: 0) 1 BIT(S) REGISTER | | | |
|---|---|---|---|
| NO. | RANGE | SOURCE | TRANSFER CONDITION |
| 26~ 1 | (0) | 0 | CPUSSTORE2 ∧ READY |
| 26~ 2 | | 1 | (CPUSIR(0:3):= 2) ∧ CPUSIF2 |

| NO. | NEXT STATE | LAST STATE | CONDITION |
|---|---|---|---|
| 40~ 1 | CPUSSTORE1 | CPUSSTORE0 | OUT |

図 5 トランスレーションリスト
Fig. 5 Translation list (SAMPLE 1).

検証するには次のようなことを確かめなくてはならない（図5のトランスレーションリストを参照する）．ターミナル ADRS について(i)の矛盾がないための条件は，

No. 1-1 と 1-2 の条件は排反． (2.1)

ただし，No. 1-1, 1-2 はトランスレーションリストのオペレーションを示す（以下同様）．

オートマトン MS のステート IDLE について(ii)，(iii)の矛盾がないための条件は，

No. 12-1 と 13-1 の条件は排反， (2.2)

No. 12-1 と 13-1 の条件の論理和は 1. (2.3)

これら(2.1)～(2.3)を確かめることを，論理式が恒等的に偽であること，

(論理式)≡$\phi$， (2.4)

の検証に書き直すことは容易である．すなわち，(2.1)は

$$(\mathrm{CPU\$IF0}\wedge\mathrm{CPU\$RUN})\wedge(((\mathrm{CPU\$IR}(0:3)=1)\vee(\mathrm{CPU\$IR}(0:3)=2)\vee(\mathrm{CPU\$IR}(0:3)=3)\vee(\mathrm{CPU\$IR}(0:3)=4)\vee(\mathrm{CPU\$IR}(0:3)=5))\wedge\mathrm{CPU\$IF2})\equiv\phi, \quad (2.5)$$

(2.2), (2.3)はどちらも

$$\mathrm{START}\wedge\neg\mathrm{START}\equiv\phi. \quad (2.6)$$

このように，内部矛盾のないことの検証（さらに，後に述べる検証も）を(2.4)の検証に帰着させて統一的な検証アルゴリズムを論じようとするのが本論文の方針である．

(2.4)の検証は，$A\wedge\neg A\equiv\phi$ を初めとするブール代数の知識と，

各オートマトンは常にどれか1つのステートを取る， (2.7)

$(A=v_1)\wedge(A=v_2)\equiv\phi$，（$A$ は任意のターミナル（レジスタ）；$v_1, v_2$ は異なる値） (2.8)

という知識等を使って行う．(2.5)は，CPU$IF 0 と CPU$IF 2 がオートマトン CPU の2つのステートであることから，(2.7)の知識を用いて検証される．(2.6)は自明である．1つのオートマトン内の問題の多くはこのようにして検証ができる．

ところが，ターミナル DATA について(i)の矛盾のないことを検証しようとすると，

$$\mathrm{CPU\$STORE1}\wedge\neg\mathrm{MODE}\wedge\mathrm{MS\$READ}\equiv\phi \quad (2.9)$$

の検証に帰着されるが，CPU$STORE 1 はオートマトン CPU のステート，MS$READ はオートマトン MS のステートであり，(2.7), (2.8)の知識等だけからは検証できない．これは2つのオートマトンにまたがる場合である．オートマトンごとに設計者が変わることも多く，オートマトン間のインタフェースには設

計ミスが多い．このため，オートマトン間のインタフェースの検証は重要であり，本検証方式の特徴となっている．次章でそのアルゴリズムを述べるが，各オートマトンは同一のクロックで同期しているという仮定をする．

## 3. 時間を遡るアルゴリズム

複数のオートマトンにまたがる検証等の場合には，(2.7)，(2.8) 等の知識だけから (2.4) を示すことができないことがある．そのような場合には，我々は，ある時刻において (2.4) の左辺の論理式が真となるという仮定から出発し，過去の履歴にまで遡って矛盾を導き出そうとする．矛盾が導き出せれば，背理法によって (2.4) の検証が完了する．

ところが，すべての履歴を持って時間を遡るのは得策でない．例えば，ある時刻に1ビットのレジスタ $R$ がセットされている（$R$ の値が1である）ためのその1時刻前での必要十分条件は，レジスタは記憶機能を持っているため，

（$R$ がセットされる条件）

$\vee((R=1)\wedge\neg$（$R$ がリセットされる条件））

で与えられる．このように過去の必要十分条件を追っていくと，扱うべき論理式が時間を遡るにつれて急激に大きくなって処理できなくなる場合が多い．しかも，その情報の大部分は検証にとっては無用である．そこで，必ずしも必要十分条件を追っていくことをせず，適当に必要条件に置き換えていく（必要条件に落すとは言え，誤りを正しいと検証することはない）．ステート表現が検証にとって重要な場合が多いと考えられるため，ステートについては必要十分条件（トランスレーションリストで与えられる）を追っていく．そのような2つのアルゴリズムを提案する．また，ループに陥った場合の処理を示す基本的な定理も述べる．

以下では，2つのオートマトン（$A_1, A_2$ とする）に関する検証を扱う，$A_1, A_2$ のステート全体の集合をそれぞれ $V_1, V_2$ とし，st 1$\wedge$st 2 (st 1$\in V_1$, st 2$\in V_2$) を $V_1\times V_2$（直積集合）の要素 (st 1, st 2) ともみることにする．アルゴリズムは，

$$v\wedge C \quad (v\in V_1\times V_2,\ C\text{は論理式}) \tag{3.1}$$

という形の論理式が恒等的に偽であること，

$$v\wedge C\equiv\phi \tag{3.2}$$

を示すものである．

### 3.1 置　換

時間を遡る操作は，論理式の上では，論理式に含まれる各条件を適当な論理式で置換していくことにより実現される．置換の対象となる条件には，ターミナル条件，レジスタ条件，ステート条件の3種類ある．

ターミナル条件は，ターミナル（システム外部からの外部ターミナルは除く）を含む条件である．ターミナル置換は，ターミナル条件に現れるターミナルにそのソースに関する式を代入し，そのターミナル条件と等価な論理式（ターミナル条件の同時刻における必要十分条件）を得る処理である．例えば，図3のターミナル $T$ についてのターミナル置換とは，ターミナル条件 $T$（値として0と1を取る）から $(A\wedge B\wedge C1)\vee(D\wedge C2)$ を得る処理である．ターミナル置換を行っても時間を遡ったことにはならないが，時間を遡るためには，記憶機能を持たないターミナルが論理式の中に存在してはならないため，ターミナル置換を繰り返して論理式の中のターミナル条件を除かなくてはならない（すべての記憶素子は明白に宣言されていてターミナルのループは存在しないと仮定する）．

レジスタ条件は，レジスタのみから成る条件である．レジスタ置換は，レジスタ条件に現れる（指定された）レジスタにそのソースに関する式を代入し，そのレジスタ条件と等価な1時刻前における論理式（レジスタ条件の1時刻前における必要十分条件）を得る処理である．例えば，図4のレジスタ $R$ についてのレジスタ置換とは，レジスタ条件 $R$（値として0と1を取る）から $(A\wedge C1)\vee(B\wedge C2)\vee(R\wedge\neg(C1\vee C2))$ を得る処理である．

ステート置換は，ステートにそのステートに転移するための必要十分条件を代入する処理である．レジスタ置換とステート置換によって，論理式の上で，1時刻時間を遡ることが実現できる．

### 3.2 アルゴリズムI

初めに，ステートについてだけ時間を遡るアルゴリズム（[ステップ 0]〜[ステップ 3]）を述べる．

[ステップ 0] 検証の対象である論理式 (3.1) が真であると仮定する．続いて [ステップ 1] へ．

[ステップ 1] ターミナル置換を繰り返してターミナル条件を除去する．(得られた論理式)$\equiv\phi$ ならば検証は完了する．そうでなければ [ステップ 2] へ．

[ステップ 2] ステート条件以外の条件を無条件に1（真）にする．この結果，

$$v_1\vee v_2\vee\cdots\vee v_m\ (v_1, v_2, \cdots, v_m\in V_1\times V_2)$$

という形の論理式（これを「標準形」と呼ぶ）が得られる．[ステップ 3] へ．

[ステップ 3] 論理式に現れるすべてのステートについてステート置換を行う．(得られた論理式)≡$\phi$ ならば検証は完了する．そうでなければ [ステップ 1] へ戻る．

以上がアルゴリズム I である．[ステップ 2] で必要条件に置き換え，[ステップ 3] で1時刻遡っている．

論理式 $C_1$ に対して [ステップ 1]〜[ステップ 3] を適用して得られる論理式を $\mathrm{np}(C_1)$ と書くことにする (necessary pre-condition)．$\mathrm{np}(C_1)$ は，ある時刻で $C_1$ が成り立つためにはその1時刻前において成立していなければならない必要条件になっている．さらに，

$$\mathrm{np}^i(C_1)=\mathrm{np}(\mathrm{np}^{i-1}(C_1)) \quad (i\geq 1) \tag{3.3}$$

$$\mathrm{np}^0(C_1)=C_1 \tag{3.4}$$

と定義すれば，$\mathrm{np}^i(C_1)$ は，ある時刻で $C_1$ が成り立つためにはその $i$ 時刻前において成立していなければならない必要条件である．$\mathrm{np}^i(C_1)\equiv\phi$ が示されると，ある時刻で $C_1$ が成立するという仮定は棄却される．したがって，アルゴリズム I によって (3.2) の検証ができるのである．

アルゴリズム I の実行に伴い，$V_1\times V_2$ の要素および $\phi$ を頂点の内容とするグラフ（木）を構成できる．[ステップ 0] に対応して $v(\in V_1\times V_2)$ を根とし，$np(v)$ の標準形を $v_1\vee v_2\vee\cdots\vee v_m$ とした時，$v_1, v_2, \cdots, v_m$ をそれぞれの内容とする $m$ 個の頂点を根の子とし，以下同様に，「最新」の頂点（葉）$\bar{v}(\neq\phi)$ に対して，$np(\bar{v})$ の標準形に現れる $V_1\times V_2$ の各要素（$np(\bar{v})\equiv\phi$ なら $\phi$）を内容とする頂点を頂点 $\bar{v}$ の子として設ける．アルゴリズム I は検証が成功した場合のみ停止するものであったから，ここでこの木を用いてアルゴリズムの停止規則を与えておく．

[停止規則 1] すべての葉の内容が $\phi$ になれば検証は完了する（成功）．

[停止規則 2] 1つの道の上に $V_1\times V_2$ の要素が重複して現れたら，アルゴリズム I による検証は不可能であるから停止する（失敗）．

1つの道の上に $V_1\times V_2$ の要素，例えば $v$，が重複して現れたとする．アルゴリズム I による検証には $v\equiv\phi$ を示すことが必要だが，その問題が自分自身に帰着してしまっていることになる．したがって，アルゴリズム I では検証できない．これが [停止規則 2] の意味である．[停止規則 2] によって停止した場合にも，反例に関する有益な情報（どのような場合に (3.2) が成立しない可能性があるか）を設計者に提供できる．設計者は，その情報をもとに，実際に反例が存在するかどうかを検討する．

アルゴリズム I の「大きさ」については次の命題が成立する．

**命題** たかだか $|V_1|\cdot|V_2|$（両オートマトンのステートの数の積）時刻遡ればアルゴリズム I は停止する．

(証明) $|V_1|\cdot|V_2|(=|V_1\times V_2|)$ 時刻遡ってもその上に $\phi$ が現れない道があるなら，その上には $|V_1\times V_2|+1$ 個の $V_1\times V_2$ の要素が並ぶから，重複して現れる要素がある．したがって，[停止規則 2] によって停止する． (証明終)

アルゴリズム I による検証の例として (2.9) の検証を行おう．検証の際には，図5のトランスレーションリストが参照される．

[ステップ 0]

$$\mathrm{CPU\$STORE\,1}\wedge\neg\mathrm{MODE}\wedge\mathrm{MS\$READ}. \tag{3.5}$$

[ステップ 1] 図5の No. 3-1 より，

$$\mathrm{CPU\$STORE\,1}\wedge\neg\mathrm{CPU\$STFL}\wedge\mathrm{MS\$READ}. \tag{3.6}$$

[ステップ 2] $\neg$CPU$ を1にして，

$$\mathrm{CPU\$STORE\,1}\wedge\mathrm{MS\$READ}. \tag{3.7}$$

[ステップ 3] No. 14-1, 40-1 より，

$$\mathrm{CPU\$STORE\,0}\wedge\mathrm{OUT}\wedge\mathrm{MS\$RCY}\wedge(\mathrm{CTR}\,(0:2)=1). \tag{3.8}$$

[ステップ 1] No. 4-1 より，

$$\mathrm{CPU\$STORE\,0}\wedge\mathrm{MS\$READ}\wedge\mathrm{MS\$RCY}\wedge(\mathrm{CTR}\,(0:2)=1). \tag{3.9}$$

(2.7) の知識 MS$READ∧MS$RCY≡$\phi$ によって (3.9)≡$\phi$ がわかるから (2.9) の検証は完了する．CPU・MS 間のターミナル（双方向バス）DATA については前章(i)の矛盾はない．

アルゴリズム I によってこのように簡単に検証できたのは，(2.9) がオートマトン CPU のステート STORE 1 とオートマトン MS のステート READ の排反性に基づき，しかも CPU・MS 間の同期をターミナル OUT で取っていたためである（**図 6**）．ターミナルの信号でオートマトン間の同期を取ることは多いから，アルゴリズム I が有効なケースは多いと考えられる．

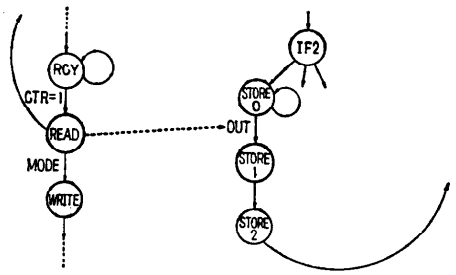



図 6 ステートダイアグラム
Fig. 6 State diagram (SAMPLE 1).

### 3.3 アルゴリズム II

オートマトン間の同期がレジスタによって取られている時等，レジスタについても時間を遡らなければ検証できない場合がある．また，明らかにレジスタについても時間を遡ることを要求される場合もある．このような場合のアルゴリズムとして，レジスタについても時間を遡るアルゴリズムを次に述べる．ただし，設計者には検証に関係するレジスタは識別できるであろうから，あらかじめ設計者が指定したレジスタについてだけ時間を遡ることにする．このアルゴリズム([ステップ 0]～[ステップ 3]) はアルゴリズム I をより精密にしたものである．

[ステップ 0] と [ステップ 1] はアルゴリズム I と同じ．

[ステップ 2] 指定されたレジスタのみから成るレジスタ条件およびステート条件以外の条件を無条件に 1 (真) にする．[ステップ 3] へ．

[ステップ 3] 論理式に現れるすべてのステート条件および指定されたレジスタのみから成るレジスタ条件についてステート置換，レジスタ置換を行う．(得られた論理式)$\equiv\phi$ ならば検証は完了する．そうでなければ [ステップ 1] へ戻る．

(アルゴリズム II 終)

アルゴリズム I の場合と同様に，論理式 $C_1$ に対して [ステップ 1]～[ステップ 3] を適用して得られる論理式を $\mathrm{np}(C_1)$ と書き (一般にこの $\mathrm{np}(C_1)$ はアルゴリズム I の $\mathrm{np}(C_1)$ より強い条件である)，(3.3)，(3.4) により $\mathrm{np}^i(C_1)$ を定義すれば，やはり，$\mathrm{np}^i(C_1)$ は，ある時刻で $C_1$ が成り立つためにはその $i$ 時刻前において成立していなければならない必要条件である．したがって，アルゴリズム I の場合と同じ理由で，アルゴリズム II によって (3.2) の検証ができるのである．

### 3.4 ループに陥った場合の処理

時間を遡っていった時得られる論理式 $C(\not\equiv\phi)$ に対して，

$$\mathrm{np}(C)=C\vee C_1 \quad (C_1 \text{ は論理式})$$

であると，

$$\mathrm{np}^i(C)=C\vee C_i \quad (i=1,2,3,\cdots;\ C_i \text{ は論理式}) \tag{3.10}$$

となり，$\mathrm{np}^i(C)\equiv\phi$ となる $i$ は見い出せない．検証は不可能にみえる．

時間を遡っていった場合，しばしばこのようなループに陥る．ループの処理を示すのが次の定理である．

[定理] 論理式 $C_1$ に対して，

$$\mathrm{np}(C_1)=(C_1\wedge C_2)\vee C_3 \quad (C_2, C_3 \text{ は論理式})$$

が成り立ち，「$C_3\wedge C_1\wedge C_2\equiv\phi$ を満たす論理式 $C_3$ が過去の少なくとも 1 時刻において成立している」という表明を設計者が与えることができる時，$C_3\equiv\phi$ を示すことができるならば，$C_1\equiv\phi$ である．

(証明) $C_3\equiv\phi$ だから，$\mathrm{np}(C_1)=C_1\wedge C_2$.

$$\mathrm{np}^2(C_1)=\mathrm{np}(C_1\wedge C_2).$$

np の定義から $\mathrm{np}(C_1\wedge C_2)\supset\mathrm{np}(C_1)\wedge\mathrm{np}(C_2)$ だから，

$$\mathrm{np}^2(C_1)\supset\mathrm{np}(C_1)\wedge\mathrm{np}(C_2)=C_1\wedge C_2\wedge\mathrm{np}(C_2).$$

さらに，

$$\mathrm{np}^i(C_1)\supset C_1\wedge C_2\wedge\mathrm{np}(C_2)\wedge\cdots\wedge\mathrm{np}^{i-1}(C_2) \quad (i=1,2,3,\cdots).$$

$C_3\wedge C_1\wedge C_2\equiv\phi$ だから，

$$\mathrm{np}^i(C_1)\supset\neg C_3 \quad (i=1,2,3,\cdots).$$

したがって，$C_1$ の成立を仮定すると，設計者の与えた表明に矛盾する．

$\therefore\ C_1\equiv\phi.$ (証明終)

定理の条件が満たされれば，$C_1\equiv\phi$ の検証が $C_3\equiv\phi$ の検証だけに帰着されるというわけである．

本章では，時間を遡って検証を行うアルゴリズムを示した．大規模システムを対象とする検証の重要な部分を占めるインタフェースの検証がこのアルゴリズムにより可能となる．

このアルゴリズムは，完全に必要十分条件を追って時間を遡るわけではないから，落してしまった情報のために検証が不成功に終わることも起こる．情報の選択が重要な問題であり (どんな選択によっても誤りを正しいと検証することがないことは言うまでもないが)，それには設計についての知識が要るともいえる．しかし，検証が不成功に終わった場合でも，今度はどのような検証を行えばよいかに関する情報が得られるから，それに基づいて検証をやり直すことができる．

## 4. 基本仕様を満たしていることの検証

本章では，機能設計が基本仕様を満たしていることの検証について述べる．この種の検証も，基本仕様（の断片）を論理式の形で与えることにより，前述のアルゴリズムⅠあるいはアルゴリズムⅡを用いる検証に帰着できることを，1つの例を通して明らかにする．まず，検証の例として設計する計算機の基本仕様を示し，それに基づく計算機のモデルの DDL による設計を与え，その検証を (2.4) の検証に帰着させる．(2.4) の検証がアルゴリズムⅡによって行われ，基本仕様を満たしていることが検証される．

### 4.1 基本仕様と機能設計

これから検証の例として設計するものはストアスルー方式の計算機である．現在の大型計算機には CPU（中央処理装置）内にバッファストレジ（主記憶装置と CPU の速度差を調整するための小容量の高速記憶装置）を持つものが多い，ストアスルー方式では，チャネル（入出力装置と記憶装置の間でデータの授受を行うための装置）から転送される新しいデータは主記憶装置にだけ格納される（対応する古いデータがバッファストレジにあっても書き直さない）．CPU がバッファストレジの古いデータをフェッチ（取り込み）しようとすると，主記憶装置からバッファストレジへのデータの取り込み (Move-in) が起こる．このようなストアスルー方式では，CPU が常に最新のデータを使うように設計されていなくてはならない．

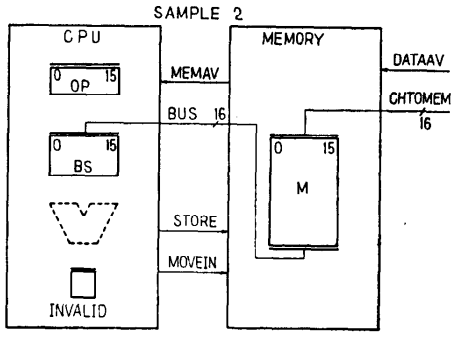



図 7 ブロックダイアグラム
Fig. 7 Block diagram (SAMPLE 2).

```
<SYSTEM> SAMPLE2:
  <TIME> P(100).    /* CLOCK */
    <ENTRANCE> DATAAV,CHTOMEM(16). /* TERMINALS FROM OUTSIDE */
    <REGISTER> OP(16).
    <TERMINAL> MEMAV,BUS(16),STORE,MOVEIN.
    <AUTOMATON> CPU: P: /* AUTOMATON CPU IS CONTROLLED BY CLOCK P */
      <REGISTER> BS(16),INVALID.
      <STATES>
        A:          /* STATE A */
          | MEMAV |
            | OP(0:1):=0 | /* CHANNEL STORE */
              INVALID←1,STORE←1,→A..
            | OP(0:1):=1 | /* FETCH */
              | INVALID | /* MOVE-IN */
                MOVEIN←1,→B
              : →A...     /* FETCH OPERATION IS NOT DESCRIBED */
            | ¬(OP(0:1):=0) ∧ ¬(OP(0:1):=1) |
              /* OPERATIONS ARE NOT DESCRIBED EXCEPT STORE AND FETCH */
              →A.
          : →A..
        B:          /* STATE B */
          INVALID←0,BS←BUS,→A.
      <END>.
    <END> CPU.
    <AUTOMATON> MEMORY: P: /* AUTOMATON MEMORY IS CONTROLLED BY CLOCK P */
      <REGISTER> M(16).
      <STATES>
        D:          /* STATE D */
          MEMAV=1,
          | STORE |  →E..
          | MOVEIN |  →F..
          | ¬STORE ∧ ¬MOVEIN | →D..
        E:          /* STATE E */
          | DATAAV |  /* CHANNEL STORE */
            M←CHTOMEM,→D
          : →E..
        F:          /* STATE F */
          BUS←M,→D.  /* MOVE-IN */
      <END>.
    <END> MEMORY.
<END> SAMPLE2.
```

図 8 DDL による機能設計
Fig. 8 Functional design in DDL.

CPU は常に最新のデータを使う，ということを仕様（の一部分）とする計算機のモデルの設計の例 SAMPLE 2 のブロックダイアグラム，DDL による機能設計，トランスレーションリストを図 7，図 8，図 9 に示す．簡単のため，主記憶Mとバッファ BS はどちらも1ワードとし，CPU の行う処理もチャネルストア（チャネルから主記憶へのデータの転送）とバッファのデータのフェッチ以外記述していない（命令は外部からレジスタ OP に転送されるものとする）．チャネルストアがあると，1ビットレジスタ INVALID をセットしてバッファの内容を無効にする．INVALID が ON の時フェッチがあると，Move-in を行い INVALID をリセットする．CPU はこれらの処理を逐次行うから，図8の設計は仕様を満たしていると考えられる．このことを検証しよう．

フェッチの処理は CPU のステートが $A$，ターミナル MEMAV の信号が ON，レジスタ INVALID が OFF の時行われる．したがって，$A \wedge$MEMAV $\wedge \neg$INVALID が成り立っている時には BS=M が成立していなければならない．そこで，我々は仕様を

$$(A \wedge \text{MEMAV} \wedge \neg \text{INVALID}) \supset (\text{BS}=\text{M}) \tag{4.1}$$

という論理式で表現することができる．図8の設計が仕様を満たしていることの検証が (4.1) の検証に帰着された．我々は，この代わりに，その否定を取った論理式が恒等的に偽であること，

$$A \wedge \text{MEMAV} \wedge \neg \text{INVALID} \wedge (\text{BS} \neq \text{M}) \equiv \phi, \tag{4.2}$$

の検証を行う．このようにして，基本仕様を満たしていることの検証を(2.4)の型(4.2)の検証に帰着す

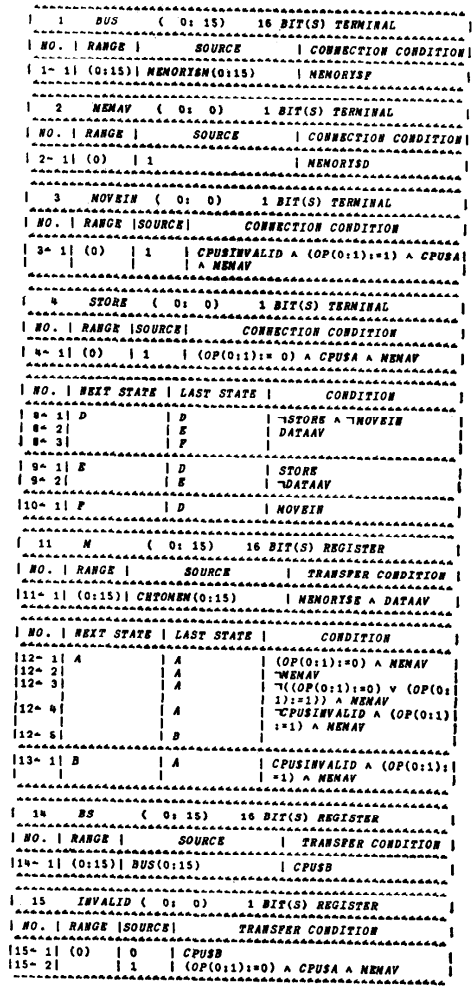



図 9 トランスレーションリスト
Fig. 9 Translation list (SAMPLE 2).

ることができた.

### 4.2 アルゴリズム II による検証

ここでは，レジスタの中で INVALID, BS, M については時間を遡ることにしてアルゴリズム II を適用し，(4.2)の検証を行う．検証の際には，図9のトランスレーションリストが参照される．

[ステップ 0]

$$A\wedge\text{MEMAV}\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{M}). \tag{4.3}$$

[ステップ 1] 図9の No. 2-1 より，

$$A\wedge D\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{M}). \tag{4.4}$$

[ステップ 3]

$$①\vee②\vee⑧\vee④\vee⑤\vee⑥. \tag{4.5}$$

ただし，①～⑥は以下の論理式である．

$$①\equiv A\wedge D\wedge\neg(\text{OP}:=0)\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{M})$$

$$②\equiv A\wedge E\wedge\text{DATAAV}\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{CHTOMEN})$$

$$③\equiv A\wedge F\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{M})$$

$$④\equiv B\wedge D\wedge(\text{BUS}\neq\text{M})$$

$$⑤\equiv B\wedge E\wedge\text{DATAAV}\wedge(\text{BUS}\neq\text{CHTOMEM})$$

$$⑥\equiv B\wedge F\wedge(\text{BUS}\neq\text{M}).$$

BS が BUS で，M が CHTOMEM で置き換わった箇所があるのは，それぞれ No. 14-1, No. 11-1 のオペレーションによる．

一方，アルゴリズム I を用いて，$A\wedge F\equiv\phi$, $B\wedge D\equiv\phi$, $B\wedge E\equiv\phi$ が示せるから，③, ④, ⑤$\equiv\phi$. したがって，論理式 ①∨②∨⑥ に対して処理を続ける．

[ステップ 1] No. 1-1 より，

$$①\vee②\vee(B\wedge F\wedge(\text{M}\neq\text{M}))\equiv①\vee②. \tag{4.6}$$

ここで，(4.4)を考え合せると，

$$C_1\equiv A\wedge D\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{M}),$$

$$C_2\equiv\neg(\text{OP}:=0),$$

$$C_3\equiv A\wedge E\wedge\text{DATAAV}\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{CHTOMEM}),$$

$$C_4\equiv\text{INVALID},$$

として前章の定理が適用できることがわかる．処理開始の時点では，バッファは空でありレジスタ INVALID が ON になっているのが普通である（この設計では簡単のためこの処理は省略してある）ので，設計者が，過去の少なくとも1時刻において INVALID が成立しているという表明を与えることができるからである．定理によって，(4.2)の検証は結局

$$A\wedge E\wedge\text{DATAAV}\wedge\neg\text{INVALID}\wedge(\text{BS}\neq\text{CHTOMEM})\equiv\phi \tag{4.7}$$

の検証に帰着される．

[ステップ 2] $A\wedge E\wedge$DATAAV$\wedge\neg$INVALID$\wedge$(BS$\neq$CHTOMEM) に適用して，

$$A\wedge E\wedge\neg\text{INVALID}. \tag{4.8}$$

[ステップ 3]

$$(A\wedge E\wedge\neg\text{DATAAV}\wedge\neg\text{INVALID})\vee(B\wedge E\wedge\neg\text{DATAAV}) \tag{4.9}$$

ここで，再び定理を適用することができ（やはり $C_4\equiv$INVALID としてよい），$B\wedge E\equiv\phi$（アルゴリズム I から）と合わせて，

$$A\wedge E\wedge\neg\text{INVALID}\equiv\phi$$

が得られる．したがって，(4.2)は検証された．

$\therefore$ $(A \wedge \mathrm{MEMAV} \wedge \neg \mathrm{INVALID}) \supset (\mathrm{BS}=\mathrm{M})$.

これで，図8の設計が仕様（の一部分）を満たしていることが検証できた．

この例から明らかなように，設計者が仕様を論理式の形で与えることができれば，(2.4) の検証に帰着させ，前章のアルゴリズムを適用することができる．

## 5. 実験システムと実用システム

ミニコンピュータの上に，(2.7), (2.8) の知識を用いて (2.4) を検証する会話形実験システムを作った．命題論理において論理式がトートロジであるかどうかを判定するアルゴリズムを中心に，(2.7), (2.8) の知識を参照できるように構成されたソフトウェアである．これを用いて，DDL で記述された大型計算機の機能設計を調べた．CPU の 30 K ゲート程の部分の機能設計であり，DDL の記述量は 10 K ステップ程，4つのオートマトンを持ち，それぞれのオートマトンは8個前後のステートを持っている．この DDL プログラムを大型計算機 (M-180 II) の上で動いている DDL トランスレータによりトランスレーションした結果について，1つのターミナル（レジスタ）に異なる値を結合(転送)する条件のペア 228 個に第2章(i)の矛盾がないことを調べた．その分類結果を図 10 に示す：

(1) (2.7)により検証できるもの．

(2) (2.8)により検証できるもの．

(3) $B \wedge \neg B \equiv \phi$ (B は論理式) により検証できるもの．

(4) 複数のオートマトンにまたがり，時間を遡る検証を使わなければならないもの．

(4)は，少数ながら，その検証は重要で困難である．実験システムでは，(4)の半数の検証には成功しているが，不明のものも残っている．それは，設計者が，

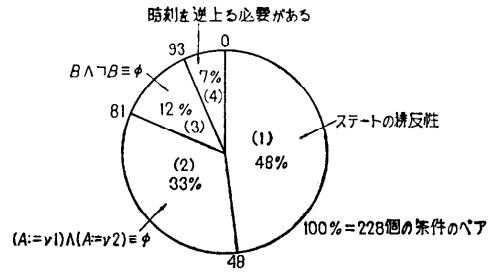



図 10 排反性の分類
Fig. 10 Classification of exclusiveness.

並列に行われる複数の処理に要する時間の長短を根拠として設計した箇所である．このような並列的な処理の部分やパイプライン的な処理の部分についての検証は，これまでに述べた手法の射程内にあると思われるが，小規模な実験システムでは，記憶容量の大きさ等に問題がある．実用的なシステムを作る場合には，論理式の解析に伴うリスト処理のために，ある程度の大きさの記憶容量を使え，リスト処理がうまく行えることが不可欠である．また，論理式全体について一挙に処理するのではなく，第3章に示した実行木に関して深さ優先の処理をしていけば，記憶場所の節約になろう．また，トランスレーションの結果を効率よく格納したデータベースを作り，条件の置換の際には，代入する論理式が高速に得られるようにすることも必要であろう．

## 6. むすび

本論文では，状態遷移表現に基づいたハードウェアの検証法を提案した．この手法は，既に開発されている DDL トランスレータによって集約された設計情報を利用することによって，大型コンピュータの設計の検証にも適用できることが示された．また，特に設計ミスの多いユニット間のインタフェースの検証に有効なアルゴリズムを与えた．

現在，会話形実験システムを用いてその評価を行っているが，シミュレーションでは検出しにくい設計ミスの発見や，設計者の予期しなかった状況が起った際の原因解明に有効性を発揮していくと考えられる．今後，これらの実験結果に基づき，実用システムの開発，さらに，発見的手法の研究を進めたい．

最後に，日頃ご指導頂く情報処理研究部宮川部長，有益なご助言・ご討論をして頂いた上原第一研究室長，川戸氏，斉藤氏始め研究室の方々，また本研究の動機づけと実際面からの御教示を頂いた槌本部長代理，徳倉課長を始めとする設計者の方々に深く感謝致します．

## 参考文献

1) Duley, J. R. and Dietmeyer, D. L.: A Digital System Design Language (DDL), IEEE Trans. on Comp., Vol. C-17, No. 9, pp. 850-861 (1968).
2) Dietmeyer, D. L.: Logic Design of Digital Systems, p. 800, Allyn and Bacon (1977).
3) Kawato, N., Saito, T., Maruyama, F. and Uehara, T.: Design and Verification of Large

Scale Computer by using DDL, Proc. 16th Design Automation Conference, pp. 360-366 (June 1979).
4) Wagner, T. J.: Hardware Verification, Ph. D. dissertation, Comp. Sci. Dept., Stanford Univ., (Sep. 1977).
5) Darringer, J. A.: The Application of Program Verification Techniques to Hardware Verification, Proc. 16th Design Automation Conference, pp. 375-381 (June 1979).

(昭和54年8月31日受付)