計算機アーキテクチャ　98－8
設　計　自　動　化　65－8
（1993. 1. 21）

実行命令数に関する数学的帰納法と再帰帰納法による形式検証

庄内　亨　　清水　嗣雄
（株）日立製作所　中央研究所
〒185　東京都国分寺市東恋ヶ窪1-280
E-mail:shonai@crl.hitachi.co.jp

あらまし

プロセッサ等の順序論理回路を再帰関数としてモデル化して行なう形式的論理検証において、実行命令数に関する数学的帰納法を用いる証明法と再帰帰納法を用いる証明法を提案した。この方法では、順序論理回路をモデル化したレジスタトランスファ記述の再帰関数を、動作記述の再帰関数に変換した後、帰納法を適用する。例として、2段パイプライン制御プロセッサを3方法で証明し、比較評価したところ、従来の動作クロック数に関する数学的帰納法に比べ、実行命令数に関する数学的帰納法を用いると証明すべきケース数を約1／2に削減できた。更に、再帰帰納法を用いると証明のための書き換え操作数を減らせる。評価対象の拡大が今後の課題である。

和文キーワード　形式検証、再帰関数、動作記述、数学的帰納法、再帰帰納法

Formal Verification Using Mathematical and Recursive Induction on the Number of Instructions Executed

Toru Shonai　Tsuguo Shimizu
Central Research Laboratory,Hitachi,Ltd.
1-280, Higashi-koigakubo,Kokubunji-shi, Tokyo 185, Japan
E-mail:shonai@crl.hitachi.co.jp

Abstract

This paper describes methods using mathematical and recursive induction on the number of instructions executed for formal verification of sequential logic circuits such as processors when modeling these circuits in terms of recursive functions. Register-transfer-type recursive functions modeling sequential circuits are transformed to behavior-type recursive functions, which are then proven with induction. Comparison among three methods for proving the correctness of a 2-stage pipelined processor is given. By using the two new methods based on mathematical and recursive induction on the number of instructions executed, it is found that the number of cases necessary to be proven is reduced by half of that using mathematical induction on the number of cycles executed in the previous approach. The recursive induction method also reduces rewriting operations, further increasing verification speed.

英文 key words　formal verification, recursive function, behavioral description, mathematical induction, recursive induction.

1．緒言

LSIの高集積化はとどまるところを知らず、それに伴ってプロセッサの集積度も依然増加している。ここ数年、増加した集積度を活用して、高速プロセッサを実現するための論理方式が一段と複雑化しており、SuperscalarやVLIWなどの並列実行方式、仮想レジスタ方式、分岐予測方式などが採用され始めている。論理方式が複雑になればなるほど、論理検証の重要性は増加する。

論理検証とは、設計した論理回路・論理装置（以下、論理と呼ぶ）の中の論理不良（バグ）を検出・修正することを言い、コンピュータの開発工程の中でも重要な工程である。

論理検証には、シミュレーションを用いる方法と机上チェックによる方法と形式的証明を用いる方法とがある。

現在、一般に使われているのはシミュレーションを用いる方法と机上チェックによる方法であるが、これらは１００％完全な方法とは言えない。シミュレーションを用いる方法では、シミュレーションケースの正しさは保証できるが全てのケースをシミュレーションすることが不可能なため、論理不良がまったくないことを保証することはできない。机上チェックによる方法はシミュレーションを用いる方法と同程度かそれ以上の論理検証能力をもつが、全てのケースをチェックしたかの確認方法がなく、個別のチェック手続きの正しさの保証方法もない。

論理の正当性（正しさ）の形式的証明は机上チェックでの非形式的チェック手続きを厳密かつ正確に定義・適用し、自然数論・初等幾何学・抽象代数学などで定理を形式的に証明するように論理の正当性を形式的に証明するものであり、完全に実用化されれば論理不良がまったくないことを保証することができる。形式的証明を用いた論理検証は形式的論理検証とも呼ばれる。

形式的論理検証において、対象とする論理（プロセッサ等）の外部仕様とインプリメンテーションとの等価性を証明する際には、（１）論理を有限状態機械（FSM）としてモデル化する方法と（２）論理を再帰関数としてモデル化する方法とがある[2]。このような方法により小規模なプロセッサの会話的な証明例はある[1,2]が、実用化のためには更に（a）証明プロセスの完全自動化（b）大規模プロセッサを扱える高速証明アルゴリズムの開発等の課題を解決する必要がある。

本研究の目的は上記（２）の方法での、従来より高速な証明アルゴリズムの概要を提示することである。従来の方法では、再帰関数で表現した外部仕様とインプリメンテーションとの等価性を証明する際に、動作クロック数に関する数学的帰納法を用いていた。本手法では、（A1）動作クロック数の代わりに実行命令数に関する数学的帰納法を用いる。（A2）そのため、従来は再帰関数を用いたレジスタトランスファ（RT）記述で表現していたインプリメンテーションを、動作記述に変換し、動作クロック時刻を保存しない書換えをも用いて証明を行なう。（B）更に数学的帰納法の代わりに再帰帰納法[4,5]を用いて証明を行なう。（A1）と（A2）、又はそれに（B）を併用する方法により、従来より高速な証明処理が可能となる。

以下、２章ではコンピュータ設計の正当性の定義について述べ、３章では例題となるコンピュータの外部仕様とインプリメンテーションについて述べる。４章では動作記述を用いた証明の処理フローと例について、５章では従来手法について述べ、６章で両者を比較し、第７章で結論と今後の課題について述べる。

2．コンピュータ設計の正当性の定義

外部仕様、インプリメンテーション、及びコンピュータ設計の正当性を定義する。

外部仕様とはプログラマから見えるそのコンピュータの仕様であり、インプリメンテーションとはそのコンピュータを実現している実際に設計された論理回路・論理装置である。

［定義１：コンピュータ設計の正当性］

Ｓ，Ｆ，Ｇを以下のように定める。

Ｓ：コンピュータのリソースのうちプログラマが設定・参照可能なリソース（主記憶・プログラマブルレジスタ等の）全体をひとつと見なした時のその取り得る状態の集合。

Ｆ：そのコンピュータの外部仕様を表す（ＳからＳへの）関数。Ｆ（ｓ１）＝ｓ２（ｓ１，ｓ２はＳの要素）とはこのコンピュータが初期状態ｓ１から実行を開始すると、このコンピュータはいつかHALT命令を実行し停止して最終状態ｓ２となるという意味である。Ｆ（ｓ１）が存在しないことは、このコンピュータが初期状態ｓ１から実行を開始すると（例えば、無限ループ等で）いつまでも実行が停止しないことを意味する。

Ｇ：そのコンピュータのインプリメンテーションを表す（ＳからＳへの）関数。Ｇ（ｓ１）＝ｓ２（ｓ１，ｓ２はＳの要素）とはこのコンピュータが初期状態ｓ１から実行を開始すると、このコンピュータはいつかHALT命令を実行し停止して最終状態ｓ２となるという意味である。インプリメンテーションの中にはプログラマが設定・参照不可能なリソース（内部レジスタ）があるが、それらはコンピュータが初期設定（power-on-reset等）されたときの状態になっているとする。Ｇ（ｓ１）が存在しないことは、このコンピュータが初期状態ｓ１から実行を開始すると（例えば、無限ループ等で）いつまでも実行が停止しないことを意味する。

Ｓ，Ｆ，Ｇが上記のように定められた場合、Ｓの任意の要素ｓについて、Ｆ（ｓ）、Ｇ（ｓ）のいずれかが存在する場合にはもう一方も必ず存在し、かつ

Ｆ（ｓ）＝Ｇ（ｓ）

が必ず成り立つことを、そのコンピュータは正しいと定義する。［定義１：完］

3．例題コンピュータの外部仕様とインプリメンテーション

例題として用いる２段パイプライン構成のコンピュータlittleの外部仕様とそのインプリメンテーションについて説明する。コンピュータlittleは、簡単ではあるが、実際のパイプライン制御プロセッサの複雑さをコンパクトに再現している。

3．1 例題コンピュータの外部仕様

3．1．1 日本語と図で表した外部仕様

図1はコンピュータlittleの構造の概要を表したものである。コンピュータlittleには、ＣＰＵと２５６バイトの主メモリがある。ＣＰＵにはプログラムカウンタ(P)と演算器(ALU)がある。主メモリは1バイト単位に0－２５５でアドレス付けされている。図2にはコンピュータlittleの命令仕様を示した。コンピュータlittleには4つの命令がある。ADD命令、BGEZ命令、HALT命令、NOP命令である。

3．1．2 再帰関数で表した外部仕様

（式1）は、例題コンピュータlittleの外部仕様を、再帰関数を用いて表した記述である。関数のsyntax,semanticsはpure LISP [3]にほぼ準じた。このsyntaxでは、if文を用いた再帰関数定義法を用いて対象を記述する。

```
run(P,M) =
    if ishalt(P,M)then <P+4,M>
elseif isadd  (P,M)then
                run( P+4, M[ MM(P+2)+MM(P+3)/M(P+2) ] )
elseif isbgez(P,M) & MM(P+3)≧0 then
                run( MM(P+2), M )
elseif isbgez(P,M) & not MM(P+3)≧0 then
                run( P+4, M )
else            run( P+4, M )                          (式1)
```

ishalt(P,M) = (0002 ≦ M2(P) ≦ FFFE)　(式2)

isadd (P,M) = (M2(P) == 0000)　(式3)

isbgez(P,M) = (M2(P) == 0001)　(式4)

（式1）のrun(P,M)が外部仕様を表す関数であり、変数Pはプログラムカウンタを表し、変数Mは主メモリを表す。<P+4,M>はP+4とMとの2項組を表し、M(a)はアドレスaから始まる1バイトのメモリ内容を表し、M2(a)はアドレスaから始まる4バイトのメモリ内容を表し、M[b/a]はアドレスaから始まる1バイトのメモリ内容をbで書換えたメモリを表し、MM(a)はM(M(a))の省略記述を表す。命令をひとつずつ処理していく機構は、関数の再帰的定義を用いて表している。

一般にはhalt命令が無いコンピュータもあるが、その時には未使用のオペコードにhalt命令を割り当ててhalt命令があるコンピュータに変更してから、本手法を適用すればよい。

3．2 例題コンピュータのインプリメンテーション

つぎに、コンピュータlittleのインプリメンテーションを図3から図8を用いて説明する。このコンピュータは、実際のコンピュータが採用しているような2段パイプライン制御方式、分岐不成立予測制御方式、直前ストアと直後の命令・データとの干渉（メモリコンフリクト）制御方式を採用している。

全体はクロック発生器から出力される2相クロック（Ｔ0，Ｔ1）で制御されている。クロックはstart信号が「1」になると発生し始め、stop信号が「1」になると停止する。

主メモリ(M)、プログラムカウンタ(P)、演算器（ＡＬＵ）の他に、プログラマが設定・参照出来ないレジスタ群、デコーダ、コンフリクト検出回路がある。

ＩＲレジスタは読み出した命令を保持する命令レジスタであり、ＯＰレジスタ、Ｘレジスタ、Ｙレジスタからなる。ＣＭＤレジスタはオペコード（ＯＰ）をデコータでデコードして得られるデコード済みオペコードを保持するレジスタである。

図4と図5はＳＥＬ詳細図とデコーダ詳細図である。halt?,add?,begz?は、入力データがHALT命令のオペコードかADD命令のオペコードかBGEZ命令のオペコードかを調べる述語である（（式5）～（式7）参照）。

halt? (CMD) = (0002 ≦ CMD ≦ FFFE)　(式5)

add? (CMD) = (CMD == 0000)　(式6)

bgez? (CMD) = (CMD == 0001)　(式7)

ＺレジスタはＭＸとＭＹの加算結果を保持するレジスタである。ＴＫＮレジスタはＭＹが0以上のとき1、そうでないとき0になるレジスタである。

図7はＡＬＵ詳細図である。

コンフリクト検出器は、先行ADD命令が行うメモリへの格納が直後命令の命令そのもの又はその命令が読み出すデータを書き替える（メモリコンフリクト: M-conflict）かを検出する組み合わせ回路である。図中の式M-conflict=XE in {[PD,PD+3]+[X]+[Y]}は、ＸＥ線上のアドレス値がＰＤ以上ＰＤ＋3以下であるかまたはＸかＹと等しいときのみM-Conflict線が「1」となることを意味する。なお、[PD,PD+3]はＰＤ以上ＰＤ＋3以下の区間を表し、[X]、[Y]は[X,X]、[Y,Y]の省略記法であり、区間間の+は集合の和集合を取る演算子を表し、inは集合への包含関係を調べる述語である。

主メモリは1クロックの間に1命令読み出しと2データ読み出しと1データ書き込み（格納）を処理できるものである。ただし、書き込み領域が読み出し領域とが重なる場合には、メモリは書き込みの動作を保証するが読み出しの動作は保証せず、その保証はコンフリクト検出回路とデコード回路とＳＥＬ回路が行う。図7は主メモリの詳細図である。このメモリは実際のメモリに比べて複雑な構成になっているが、それは証明手順を見通し良くするためである。

図9は初期設定一覧表を表す。初期値が指定されていないレジスタの初期値は不定である。

つぎに、コンピュータlittleの実現での主要なステージフローを図9を用いて説明する。

コンピュータlittleの実現はIFOFステージとEXSTステージとの2ステージからなる。IFOFステージでは命令の読み出し(IF)とデータの読み出し(OF)を行い、EXSTステージでは演算(EX)と演算結果のメモリへの格納(ST)を行う。

（1）は通常ケースであり、第1から第3までの命令が2段パイプラインで整然と流れている。第4命令はHALT命令であり、そのEXSTステージ完了直後にクロックを止め、コンピュータを停止させる。

（2）は分岐成功ケースであり、第2命令が成功する分岐命令である。不成功側の次命令（第3命令）の実行は途中で中断され、成功側の次命令（第4命令）のIFOFステージの実行が分岐命令のEXSTステージ完了直後に開始される。

（3）はメモリコンフリクトケースであり、第1と第2命令の間にメモリコンフリクトがある。この場合、第1命令のEXSTステージと同時に開始される第2命令の実行は途中

で中断され、第1命令のEXSTステージ完了直後に（メモリへの格納完了直後に）第2命令の実行がIFOFステージから再開される。

4．　動作記述を用いた証明の処理フローと例

処理フローは以下の主要3処理からなる。

（処理1）構造記述からRT記述への変換

（処理2）RT記述から動作記述への変換

（処理3）帰納法による正当性の証明

帰納法としては、ここでは数学的帰納法と再帰帰納法を用いる。以下、順に説明する。

4．1 構造記述から再帰関数ＲＴ記述への変換

構造記述からＲＴ記述への変換処理では、構造記述で表現したインプリメンテーションをＲＴ記述で表現したインプリメンテーションに変換する。変換処理フローの詳細は例えば[2]に記載がある。ＲＴ（register transfer：レジスタ転送）記述とは、各レジスタの次のクロック時刻での更新値を各レジスタごとに一ヶ所にまとめてある記述である。

例題の構造記述の図3から図8をＲＴ記述に変換したのが、（式8）と（式9）である。（式10）はXD in {[P,P+3]+[M(P+2)]+[M(P+3)]}の省略記法の定義である。

```
i-run(P,M)=f(P,M,-,-,-,nop).                                    (式8)

f(P,M,XD,MX,MY,CMD)
= if halt? (CMD) then <P,M>
else f( if add? (CMD) & M-conf (XD,P,M(P+2),M(P+3)) or
        halt?(CMD)                                  then P
     elseif bgez?(CMD) & MY≧0 then MX else P+4 fi,
        if add? (CMD) then M[MX+MY/XD] else M fi,
        if {add? (CMD) & M-conf (XD,P,M(P+2),M(P+3))}
          or{bgez?(CMD) & MY≧ 0 }
          then <M(P+2), MM (P+2), MM (P+3), nop >
          else <M(P+2), MM (P+2), MM (P+3),M2(P) > fi )
fi                                                              (式9)

M-conf (XD,P,M(P+2),M(P+3))
= XD in {[P,P+3]+[M(P+2)]+[M(P+3)]}                             (式10)
```

（式9）では、

```
if p then <a,b,c,d> else <e,f,g,h> fi                           (式11)
```

の形の記法を用いたが、これは、

```
if p then a else e fi ,if p then b else f fi ,
if p then c else g fi ,if p then d else h fi                    (式12)
```

の省略記法である。

ＲＴ記述は主関数ｉ－ｒｕｎと補助関数ｆからなる。主関数は、プログラマブル・レジスタに特定の値が設定された時に最終的にプログラマブル・レジスタに得られる値を表す。主関数は、プログラマブル・レジスタはそのままにし、非プログラマブル・レジスタには初期値を設定して補助関数を呼び出す。補助関数は、プログラマブル・レジスタと非プログラマブル・レジスタに特定の値が設定された時に最終的にプログラマブル・レジスタに得られる値を表す。補助関数は、クロック停止条件が成立した時には、その時のプログラマブル・レジスタの値を最終的に得られる値として返し、そうでない時には、各レジスタを次のクロック時刻の値に更新して補助関数を再び呼び出す。各クロック時刻での各レジスタの値の更新を関数の再帰呼出しを用いて表している。

4．2 再帰関数ＲＴ記述から再帰関数動作記述への変換

ＲＴ記述から動作記述への変換処理では、ＲＴ記述で表現したインプリメンテーションを動作記述で表現したインプリメンテーションに変換する。ここで、動作記述とは、ｉｆ条件はすべてｆの外側にあり、各条件ごとの処理動作が一ヶ所にまとめてある記述である。

変換処理フローでは、ｆの内側にあるｉｆをｆの外側に移し、外側のｉｆ条件を参照しながら内側のｉｆ条件を簡単化することを、内側にｉｆがなくなるまで繰り返す。

（式9）の例では、動作記述として（式13）が得られる。

```
f(P,M,XD,MX,MY,CMD)
= if halt? (CMD) then <P,M>
elseif add? (CMD) & M-conf (XD,P,M(P+2),M(P+3)) then
     f(P, M[MX+MY/XD] ,M(P+2), MM (P+2), MM (P+3), nop )
elseif bgez?(CMD) & MY≧0 then
     f( MX, M      , M(P+2), MM (P+2), MM (P+3), nop )
elseif add? (CMD) then
  f(P+4,M[MX+MY/XD],M(P+2),MM (P+2),MM (P+3),M2(P) )
else f( P+4, M,      M(P+2), MM (P+2), MM (P+3),M2(P) )
fi                                                              (式13)
```

4．3 実行命令数に関する数学的帰納法による正当性の証明

ここでは、プロセッサｒｕｎ（Ｐ，Ｍ）が停止する(HALT命令を実行しクロックが停止される)までの実行命令数ｎに関する数学的帰納法による証明について述べる。実行命令数ｎとは（式1）におけるｒｕｎの定義本体の呼出し回数と等価である。

4．3．1 数学的帰納法による証明フロー

証明フローを図10に示す。step1,2は数学的帰納法におけるｎ＝1のケースの証明部分であり、step3,4,5は数学的帰納法におけるｎ＝mのケースを仮定してのｎ＝m＋1のケースの証明部分である。なお、Ｄ1への代入処理部では数学的帰納法におけるｎ＝mケースの仮定を使用している。また、D3,E2,E4への代入処理部では動作クロック時刻を保存しない書換えを利用している。フロー中のメモリ公理とは「書換えなしメモリデータ不変公理」の略称であり、次の式で表される。

$$s \neq f \Rightarrow M[c/s](f) = M(f).$$

左辺の条件が成立するときには右辺のような書換えが可能であり、「書換えられていないメモリデータは不変である」ということを意味している。

4．3．2 例題の証明

（式8）～（式9）と（式1）を用い、証明フローについて説明する。まずstep1ではｒｕｎ（Ｐ，Ｍ）がｎ＝1で停止する条件すなわち'halt(P,M)=true'をＡに代入し、続いてＡが真の時のｒｕｎ（Ｐ，Ｍ）の値すなわち'<P+4,M>'をＢに代入し、ｉ－ｒｕｎ（Ｐ，Ｍ）の値すなわち'<P+4,M>'を

Cに代入する。step2ではBとCの一致を確認する。

step3ではｒｕｎ（Ｐ，Ｍ）の定義本体のｒｕｎ（Ｐ，Ｍ）をｉ－ｒｕｎ（Ｐ，Ｍ）で置換した後、ｉ－ｒｕｎ（Ｐ，Ｍ）をその定義本体で置換し、ｎｏｐがある部分にｆを適用してｎｏｐを消去した式

```
run(P,M) =
 if ishalt(P,M)  then <P+4,M>
elseif isadd (P,M)
     then f(   P+4+4  ,
          M[MM(P+2)+MM(P+3)/M(P+2)]          ,
          M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+2 ) ,
          M[MM(P+2)+MM(P+3)/M(P+2)]
            M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+2 ),
          M[MM(P+2)+MM(P+3)/M(P+2)]
            M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+3 ),
          M[MM(P+2)+MM(P+3)/M(P+2)]2( P+4 )   )
elseif isbgez(P,M) & MM(P+3)≧0
     then f( (MM(P+2)) +4,
          M,
          M( (MM (P+2)) +2 ),
          MM( (MM (P+2)) +2 ),
          MM( (MM (P+2)) +3 ),
          M2( (MM (P+2)) ) )
elseif isbgez(P,M) & not MM(P+3)≧0
     then f( P+4+4  , M, M( P+4+2), MM( P+4+2 ),
            MM( P+4+3 ), M2 ( P+4 ) )
else       f( P+4+4  , M, M( P+4+2), MM( P+4+2 ),
            MM( P+4+3 ), M2 ( P+4 ) )
fi                                                    （式１４）
```

をＤ３に代入する。step4ではｉ－ｒｕｎ（Ｐ，Ｍ）の定義にｎｏｐがあるので、　ｎｏｐがある部分にｆを適用してｎｏｐを消去したのち、そのｆをその定義本体で置換した式

```
 i-run(P,M)=
 if halt? (M2(P)) then <P+4,M>
elseif add? (M2(P))
     & M-conf(M(P+2),P+4,M(P+4+2),M(P+4+3)) then
       f( P+4  , M[MM (P+2)+MM (P+3)/M(P+2)],
         M(P+4+2), MM (P+4+2), MM (P+4+3), nop )
elseif bgez?(M2(P)) & MM (P+3)≧0  then
       f( MM (P+2), M      ,
         M(P+4+2), MM (P+4+2), MM (P+4+3), nop )
elseif add? (M2(P))  then
       f( P+4+4, M[MM (P+2)+MM (P+3)/M(P+2)],
         M(P+4+2), MM (P+4+2), MM (P+4+3),M2(P+4) )
else     f( P+4+4, M,
         M(P+4+2), MM (P+4+2), MM (P+4+3),M2(P+4) )
fi                                                    （式１５）
```

をＥ３に代入する。ｎｏｐがある部分にｆを適用してｎｏｐを消去した後、メモリ公理が適用可能である。すなわち、（式１６）の書換えが可能である。

```
 not XD in {[P,P+3]+[M(P+2)]+[M(P+3)]}
        ⇒ M(P+2) = M[MX+MY/XD](P+2),
          MM(P+2) = M[MX+MY/XD]M[MX+MY/XD](P+2),
          MM(P+3) = M[MX+MY/XD]M[MX+MY/XD](P+3),
          M2(P)  = M[MX+MY/XD]2(P).          （式１６）
```

これを使って変換・統合するとE5としてD3と同じものが得られ、Ｄ３とＥ５の一致が確認される。したがって数学的帰納法により等価性が証明される。

４．４　実行命令数に関する再帰帰納法による正当性の証明

４．４．１　再帰帰納法による証明フロー

図１１は再帰帰納法を用いた証明フローである。

step1は上位の制御部分であり、step2～９の下位の処理から適用可能な処理があるかを上から順に調べ、適用可能な処理を１つ選び実行するという処理を反復実行する。step2～5で動作記述ｆを書換え、その結果を使ってstep６～８で動作記述ｉ－ｒｕｎを書換え、step９で再帰帰納法[4,5]を適用してｉ－ｒｕｎとｆの間で等価性を証明する。

再帰帰納法とは、「２つの再帰関数が関数名の相違を除いてはまったく同一であるならば、両者は等しい。」という性質を利用して再帰関数間の等価性を証明する方法である。

４．４．２　例題の証明

（式８）～（式９）と（式１）を用い、証明フローについて説明する。まず（式９）の動作記述ｆのｎｏｐがある部分にｆを適用してｎｏｐを消去する。その結果が（式１７）である。

```
 f(P,M,XD,MX,MY,CMD)
 = if halt? (CMD) then <P,M>
elseif add? (CMD) & M-conf (XD,P,M(P+2),M(P+3)) then
       f( P+4, M[MX+MY/XD], M[MX+MY/XD](P+2) ,
                M[MX+MY/XD]M[MX+MY/XD](P+2) ,
                M[MX+MY/XD]M[MX+MY/XD](P+3) ,
                M[MX+MY/XD]2(P) )
elseif bgez?(CMD) & MY≧0   then
       f( MX+4, M, M(MX+2), MM (MX+2),
         MM (MX+3),M2(MX) )
elseif add? (CMD)  then
       f( P+4, M[MX+MY/XD], M(P+2), MM (P+2),
         MM (P+3),M2(P) )
else    f( P+4, M,       M(P+2), MM (P+2), MM (P+3),M2(P) )
fi                  （式１７）
```

（式１７）をstep3で整形したのち、step5でメモリ公理を適用し、step4でｔｈｅｎ部が同一のｉｆ－ｔｈｅｎ文を統合して（式１８）を得る。

```
   f(P,M,XD,MX,MY,CMD)
=   if halt? (CMD) then <P,M>
elseif add? (CMD) then
       f( P+4, M[MX+MY/XD], M[MX+MY/XD](P+2),
                M[MX+MY/XD]M[MX+MY/XD](P+2),
                M[MX+MY/XD]M[MX+MY/XD](P+3),
                M[MX+MY/XD]2(P) )
elseif bgez?(CMD) & MY≧0   then
       f( MX+4, M, M(MX+2), MM (MX+2),
```

```
                MM (MX+3),M2(MX) )
elseif bgez?(CMD) & not MY≧0  then
        f( P+4, M,  M(P+2), MM (P+2), MM (P+3),M2(P) )
else    f( P+4, M,  M(P+2), MM (P+2), MM (P+3),M2(P) )
fi                                                    (式１８)
```

一方、動作記述ｉ－ｒｕｎのｎｏｐがある部分にｆを適用してｎｏｐを消去（step6）する。その結果が（式１９）の３行目である。

```
 i-run(P,M)
 =f(P, M, -, -, -, nop )
 =f(P+4, M, M(P+2), MM (P+2), MM (P+3), M2(P) )
 =  if halt? (M2(P)) then <P+4,M>
 elseif add? (M2(P)) then
        f(   P+4+4  ,
          M[MM(P+2)+MM(P+3)/M(P+2)],
          M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+2 ),
          M[MM(P+2)+MM(P+3)/M(P+2)]
              M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+2 ),
          M[MM(P+2)+MM(P+3)/M(P+2)]
              M[MM(P+2)+MM(P+3)/M(P+2)]( P+4+3 ),
          M[MM(P+2)+MM(P+3)/M(P+2)]2( P+4 ) )
 elseif bgez? (M2(P)) & MM(P+3)≧0 then
        f( (MM(P+2)) +4, M, M( (MM (P+2)) +2 ),
          MM( (MM (P+2)) +2 ),  MM( (MM (P+2)) +3 ),
          M2( (MM (P+2)) ) )
 elseif bgez? (M2(P)) & not MM(P+3)≧0 then
 else    f( P+4+4  , M, M( P+4+2), MM( P+4+2 ),
          MM( P+4+3 ), M2 ( P+4 ) )
 fi                                                   (式１９)
```

```
add? (M2(P)) = ( M2(P) == 0000 ) =  isadd( P,M )     (式２０)
bgez? (M2(P))  = ( M2(P) == 0001 )= isbgez( P,M )    (式２１)
halt? (M2(P))  = ( 0002 ≦ M2(P) ≦ FFFE ) = ishalt(P,M).
                                                     (式２２)
```

その際、（式２３）が置換式として登録される。

```
 i-run(P,M) = f( P+4,M,M(P+2),MM(P+2),MM(P+3),M2(P) ).
                                                     (式２３)
```

（式１９）の３行目の動作記述ｉ－ｒｕｎはｆだけの式なので、そのｆに（式１８）の動作記述ｆを適用（step7）する。（式１９）の４行目以降の動作記述ｉ－ｒｕｎに置換式を用いた書換えが実行され（step8）、更に（式２０）～（式２２）を使ってadd?,bgez?,halt?をisadd,isbgez,ishaltに置換すると、（式２４）が得られる。

```
    i-run(P,M)
 =  if ishalt(P,M) then <P+4,M>
 elseif isadd (P,M) then
                i-run(P+4,M[MM(P+2)+MM(P+3)/M(P+2)])
 elseif isbgez(P,M) & MM (P+3)≧ 0 then
                i-run( MM (P+2) ,  M  )
 elseif isbgez(P,M) & not MM (P+3)≧ 0 then
                i-run( P + 4  ,  M  )
 else           i-run( P + 4  ,  M  )                (式２４)
```

（式２４）の動作記述ｉ－ｒｕｎと（式１）のｒｕｎの間には再帰帰納法が適用でき、ｉ－ｒｕｎとｒｕｎの間で等価性が証明される。ｉ－ｒｕｎとｒｕｎは関数名の相違を除いてはまったく同一であるからである。

５．　従来手法－ＲＴ記述を用いた正当性の証明－

ここでは従来技術での動作クロック数ｎに関する数学的帰納法による正当性証明法についてその概要を例題で示す。

まず’i-run(P,M)=run(P,M)’を直接証明するのではなく、補助命題

```
 f(P+4,M, M(P+2),MM(P+2),MM(P+3),M2(P) )=run(P,M),
 and f(P, M,-,-,-,nop )=run(P,M)                     (式２５)
```

を、プロセッサが停止する(HALT命令を実行しクロックが停止される)までのインプリメンテーションの動作クロック数ｎに関する数学的帰納法で証明する。インプリメンテーションの動作クロック数ｎとは（式９）のｆの定義本体の呼出し回数である。ｎをある値に制限すると（式２５）の第１、２等式それぞれの左辺がその制限内で必ず停止するとは限らない。以下の数学的帰納法の証明では、ｎが制限された場合その制限内で上記左辺が停止するようなＰ，Ｍの値についてのみ等号が成立すると考える。

まずｎ＝１の時を考える。（式２５）の第１等式はあきらかである。（式２５）の第２等式の左辺はｎ＝１では値が確定するようなＰ，Ｍの組は存在しない。以上よりｎ＝１の時には（式２５）が成立する。

次にｎ＝ｍ（≧２）の時の成立を仮定してｎ＝ｍ＋１の時を考える。まず、（式２５）の第１等式の左辺を考える。以下ではＰ，Ｍとして、ｍ＋１クロック後にf(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))が停止するようなＰ，Ｍを考えている。（式２５）の第１等式の左辺に（式９）を適用し、式の簡単化をした後、あとｍサイクル動作しないと停止しないはずなのでhalt? (M2(P))は偽であることを考え合わせると、

```
 f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P) )
 =f(  if add? (M2(P)) & M-conf (M(P+2),P+4,
                                M(P+4+2),M(P+4+3))
                    then P+4
  elseif bgez?(M2(P)) & MM(P+3)≧0  then MM(P+2)
                    else P+4+4  fi,
    if add? (M2(P))  then  M[MM(P+2)+MM(P+3)/M(P+2)]
                    else  M   fi,
    if {add? (M2(P)) & M-conf (M(P+2),P+4,
                                M(P+4+2),M(P+4+3))}
      or{bgez?(M2(P)) & MM(P+3)≧ 0 }  then
          <M(P+4+2), MM (P+4+2), MM (P+4+3), nop >
    else <M(P+4+2), MM (P+4+2), MM(P+4+3),M2(P+4) >
                                                fi)  (式２６)
```

となる。

（イ）add? (M2(P)) & M-conf (M(P+2),P+4,M(P+4+2), M(P+4+3))が真の時

```
f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P) )
= f( P+4 , M[MM(P+2)+MM(P+3)/M(P+2)],
```

M(P+4+2), MM (P+4+2), MM (P+4+3), nop) （式２７）

となるが、この右辺はあとｍクロック動作すれば停止するはずなので帰納法の仮定が使え

f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))

= run(P+4 , M[MM(P+2)+MM(P+3)/M(P+2)]) （式２８）

となる。一方、add? (M2(P)) & M-conf (M(P+2),P+4, , M(P+4+2),M(P+4+3))が真の時には（式１）より

run(P,M)=run(P+4 , M[MM(P+2)+MM(P+3)/M(P+2)]) （式２９）

なので、' f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P)) =run(P,M) ' が成立する。

（ロ）add? (M2(P)) & not M-conf (M(P+2),P+4,M(P+4+2), M(P+4+3))が真の時

（ハ）bgez?(M2(P)) & MM(P+3)≧0が真の時

（ニ）bgez?(M2(P)) & not MM(P+3)≧0が真の時

（ホ）（イ）～（ニ）以外の時

（ロ）～（ホ）についても同様にして、' f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))=run(P,M)' が成立することが示せる。

（イ）～（ホ）により、ｎ＝ｍ（≧２）の時を仮定するとｎ＝ｍ＋１にも' f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))=run(P,M)' が成立することがわかる。（式２５）の第２等式についても同様の方法により、ｎ＝ｍ＋１の時の成立を示せる。以上により、数学的帰納法を用いて' f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))=run(P,M), and f(P, M,-,-,-,nop)=run(P,M)' が証明できた。

この補助命題と（式１１）により容易に' i-run(P,M)=run(P,M)' が証明できる。

６．比較検討

６．１ 動作クロック数に関する数学的帰納法と命令実行数に関する数学的帰納法の比較

（１）例題を見ると、前者では f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))と f(P, M,-,-,-,nop)を扱う必要があったが、後者では動作クロック数を保存しない書き換えを導入したために、 f(P, M,-,-,-,nop)が f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))に統合されて f(P+4, M, M(P+2), MM(P+2), MM(P+3), M2(P))だけを扱えばよくなった。これにより、後者では証明すべきケースが１／２に削減できた。例題は２段パイプラインであったが、３段以上のさらに複雑なパイプラインについても証明すべきケースが削減できると思われる。

６．２ 数学的帰納法と再帰帰納法の比較

（１）前者ではｎ＝１のケースとｎ＝ｍのケースを仮定してのｎ＝ｍ＋１のケースとを証明する必要があるが、後者では両者をまとめて１度で証明できるという長所がある。

（２）例題の証明過程における各書換え操作の出現頻度を調べると次表のようになる。なお、（ ）内は図１０のフローを最適化した場合の頻度である。

	数学的帰納法	再帰帰納法
(a)i-run本体での書換え	4 （1）	0
(b) f 本体での書換え	8 （5）	4
(c)メモリ公理での書換え	1	1
(d)帰納法仮定の導入	1	0
(e)再帰帰納法置換式の導入	0	1

(d)と(e)は互いに対応すると考えられ、この部分に差はない。差があるのは(a)と(b)の頻度であり、いずれも再帰帰納法を用いた方が少ない。

７．結言

７．１ 結論

（１）再帰関数を用いたモデル化による形式的論理検証の証明方法とし、従来からの動作クロック数に関する数学的帰納法を用いる方法に代わって、実行命令数に関する数学的帰納法を用いる方法及び再帰帰納法を用いる方法を提案した。

（２）２段パイプライン制御プロセッサを例題として、各方法での証明フローを示した。

（３）例題により、実行命令数に関する数学的帰納法を用いる方法では、証明すべきケースが従来より削減でき、更に再帰帰納法を用いると証明過程における書換え操作回数が削減できることを示した。

７．２ 今後の課題

主な課題を以下に示す。

（１）３段以上のパイプライン制御プロセッサへの証明フローの適用評価

（２）提案した証明法を組み込んだ形式検証システムの開発

謝辞

初期検討時に貴重な助言をくださった鈴木敬氏をはじめ、数々のご援助をくださった（株）日立製作所中央研究所の方々に感謝の意を表します。

参考文献

[1] M.Srivas and M.Bickford,"Formal Verification of a Pipelined Micro-processor," IEEE Software,1990,pp.52-64.

[2] Dominique D. Borrione,et al., "Formal Verification of VHDL Descriptions in the Prevail Environment," IEEE Design and Test of Computers,1992,pp.42-56.

[3] J.McCarthy, et al. LISP 1.5 Programmer's Manual. M.I.T. Press, Cambridge, Mass., USA,1962

[4] 伊藤貴康," プログラム理論とその応用（2)", 情報処理、1981、pp.884-895.

[5] J.McCarthy, "Towards a mathematical science of computation,IFIP '62,1963,pp.21-28."

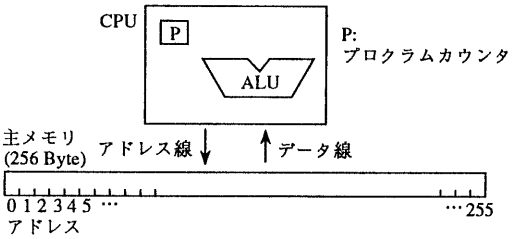
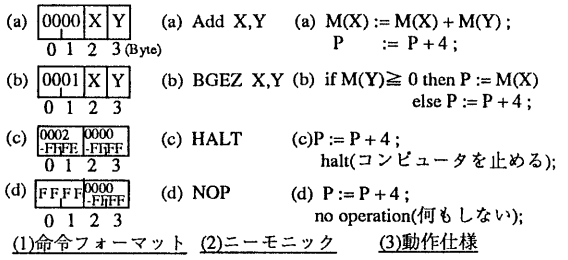


図1　コンピュータlittleの外部仕様の構造概要図



(注) M(X)はアドレスXのメモリ内容1バイトを示す。
(注) ＋は256を法とする加算を表わす。

図2　コンピュータlittleの外部仕様の命令仕様記述

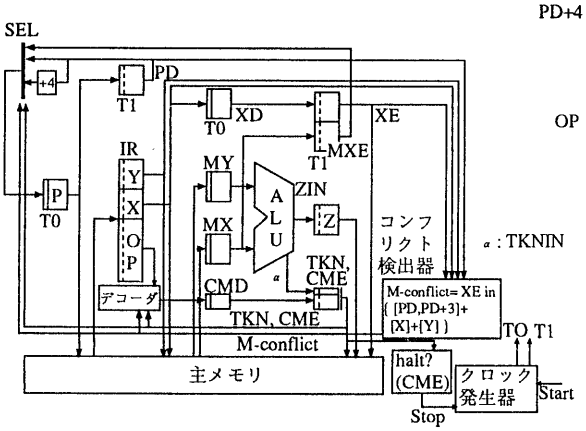


図3　コンピュータlittleのインプリメンテーションの構造図

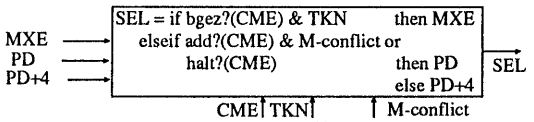


図4　SEL詳細図

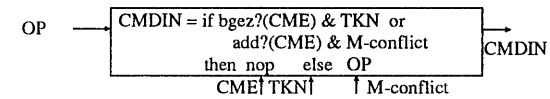


図5　デコーダ詳細図

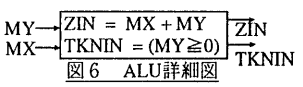


図6　ALU詳細図

リソース名	初期値
CMD	FFFF

図8　初期値一覧表

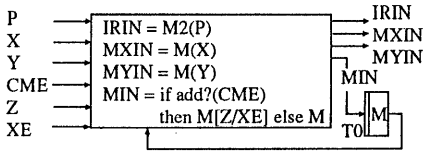
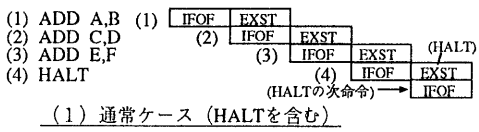
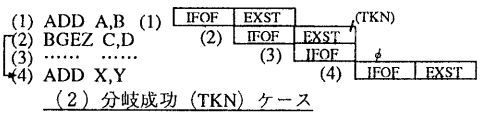


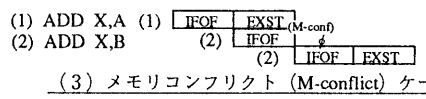
図7　主メモリ詳細図



（1）通常ケース（HALTを含む）



（2）分岐成功（TKN）ケース



（3）メモリコンフリクト（M-conflict）ケース

図9　コンピュータlittleのインプリメンテーションの主要ステージフロー図

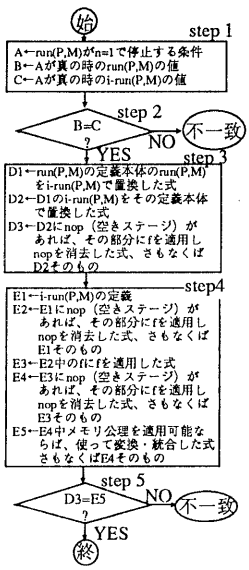


図10　数学的帰納法による証明フロー

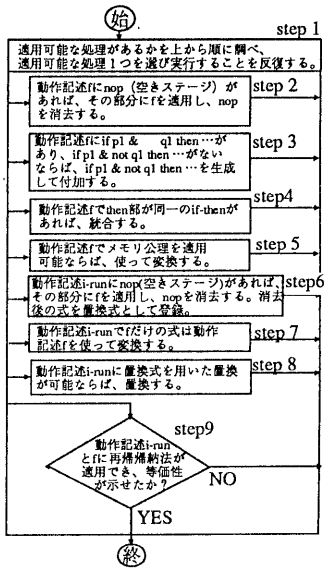


図11　再帰帰納法による証明フロー