設 計 自 動 化 81－11
(1996. 10. 18)

プレスブルガー文真偽判定手続きを用いた算術演算回路の正しさの証明

森岡 澄夫　柴田 直樹　東野 輝夫　谷口 健一

大阪大学 基礎工学部 情報工学科

〒560 大阪府豊中市待兼山町 1－3
Tel: 06－850－6607　Fax: 06－850－6609

E－mail:{morioka, n－sibata, higashino, taniguchi}@ics.es.osaka-u.ac.jp

あらまし

加算器，乗算器，ALU など，算術演算を行う組み合わせ論理回路が，そのワードレベル仕様 F（整数上の論理式として書かれた入出力関係の記述）を正しく実現している事を，プレスブルガー文真偽判定手続きを用いて自動証明する方法と，証明例について述べる．証明は，いわゆるビットレベル検証（各回路モジュール M_jごと，そのワードレベル仕様 F_jがゲートレベルで正しく実現されていることの証明）とワードレベル検証（各 M_jの接続関係および各ワードレベル仕様 F_jのもとで，Fが満たされることの証明）に分けて行う．乗算など，プレスブルガー算術で直接扱えない演算を行う回路についても，その演算に関して数学的に成り立つ性質等を仮定することにより，証明できる場合がある．本手法の特徴は，幾つかの工夫を行ったプレスブルガー真偽判定ルーチンを用いることにより，各モジュールの演算ビット長 n が増えても，回路中のモジュールの数や組合せ方が同じで，かつ仕様記述のサイズが n に依存していなければ，ワードレベル検証にかかる時間がほとんど増加しないことである．例えば n ビット乗算器から $2n$ ビット乗算器を構成した場合のワードレベル検証を，2 分程度の CPU 時間で行えた．ビットレベル検証についても，演算ビット長が 4 ビット程度であれば，例えば加減算・論理演算を行う ALU（74382）について 6 分程度の CPU 時間で行えた．

キーワード　算術演算回路，組合せ論理回路，正当性証明，階層的証明，ワードレベル検証，プレスブルガー算術

Automatic Verification of Arithmetic Circuits using a Decision Procedure for Presburger Arithmetic

Sumio MORIOKA　Naoki SHIBATA　Teruo HIGASHINO　Kenichi TANIGUCHI

Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University

Machikaneyama 1－3, Toyonaka, Osaka 560, Japan
Tel: +81－6－850－6607　Fax: +81－6－850－6609

E－mail:{morioka, n－sibata, higashino, taniguchi}@ics.es.osaka-u.ac.jp

Abstract

In this paper we present a method to prove automatically that given arithmetic circuits, such as adders, multipliers and ALUs, correctly implement their overall circuit functionality F, using Presburger arithmetic. The F is given as word-level specifications, that are the descriptions of relations between circuits' inputs and outputs on integer variables and integer operators. We adopt a hierarchical approach to verify arithmetic circuits; first the gate-level component modules M_j in the circuits are shown to implement their word-level specifications F_j (so called "bit-level verification"), and then F is proved on both F_js and the description of connections among all modules (so called "word-level verification"). The feature of our proof method is that the CPU time needed for the word-level verification is almost independent of the word-length (bit number) n of the modules, if both the circuits' architecture and the size of F and F_js are also independent of n. We have recently improved the computation time of a decision algorithm for Presburger arithmetic. Using the improved decision procedure, we have carried out the word-level verification of $2n$-bit multipliers within about 2 minutes (CPU time), where each of those $2n$-bit multipliers is constructed from four n-bit multipliers. We have also carried out the bit-level verification of a 4-bit ALU (74382) within about 6 minutes.

key words　Arithmetic Circuit, Combinatorial Circuit, Correctness Proof, Hierarchical Verification, Word-level Verification, Presburger Arithmetic

1 はじめに

1.1 背景

加算器，乗算器，ALU など，算術演算を行う組み合わせ論理回路が，任意の入力に対して正しい演算結果を出力することを，機械的に証明できることが望ましい．しかし，そのような組合せ論理回路の機能検証では一般に，回路が行う演算ビット長が増大すると，テストすべき回路入力値の数が指数的に増える，という問題がある．

そこで多ビット長演算を行う回路の証明を実際上可能にするため，従来より，証明をビットレベル検証とワードレベル検証に階層化して行う手法が広く用いられている [1]．**ビットレベル検証**では，回路中の（少ないビット長の演算を行う）モジュール M_iごと，その論理回路のもとで，M_iの機能記述 F_iが満たされることを証明する．ここでF_iは，モジュール M_iの入出力を（ビット列でなく）整数変数とみなし，そのうえで入出力関係を+, ×など整数上の演算子を使って記述したものである．このように入出力を整数とみなして書いた仕様は**ワードレベル仕様**と呼ばれる．なお，M_iの実回路の記述は，**ビットレベル仕様**と呼ばれる．一方，**ワードレベル検証**では，「各モジュールを接続したとき，そのもとで回路の全体仕様 Fが満たされる」ことを，各モジュールのワードレベル仕様 F_iと接続関係の記述のもとで証明する．ここで，回路の全体仕様 Fも，ワードレベル仕様として与えられる．

本稿では，主にワードレベル検証について取り上げる．

近年，ワードレベル検証を行うための手法が幾つか提案されている．それらの従来手法は，用いる手法により，定理証明による方法 [2, 3, 4]（HOL,PVS,RRL などの定理証明系を使う）と，BMD[1] など BDD ベースの方法とに分けられる．

定理証明による方法では，基本的に，どのような演算を行う回路でも証明の対象とすることができる．ワードレベル仕様には，行う演算の定義を記述する（いろいろな定義の仕方ができるが，再帰的定義を仕様とすることが多い）．例えば乗算であれば，

$$mult(x+1, y) = mult(x, y) + x, mult(0, y) = 0,$$
$$mult(x, y) = mult(y, x)$$

などと記述したものがワードレベル仕様となる（加算+についても，必要なら定義する）．定理証明による方法では，通常，演算ビット長をパラメータ化したうえで，帰納法などを用いてシンボリックに証明を行う．その場合，証明にかかる CPU 時間が演算ビット長に依存しないという利点がある．一方，証明作業の自動化が難しく，証明作業を定理証明系と対話的に進めるため，全体としての証明作業には手間がかかるという問題点がある．ただし，検証の対象とする演算や，その処理方式を限定すれば，証明手順を固定でき，自動で証明できる場合もある [4]．

BDD ベースの方法では，表現形式（BMD,MTBDD など）によって扱えるデータタイプや演算が限られており，その拡張は難かしい．例えば BMD の場合，検証できる回路は，整数演算のみを行い，かつ出力関数が入力に対する liner function であるものに限られる．ワードレベル仕様には，行う演算の定義を，用いる表現形式で使用できるデータタイプおよび演算を組み合わせて書く．例えば BMD では，2 ビット乗算の仕様は，

$$2x_1(2y_1 + y_0) + x_0(2y_1 + y_0)$$

のように，ビットレベルの乗算・加算に展開した形で与えられる．証明は，完全自動で，かつかなり高速に行える．例えば文献 [1] では，BMD を用い，256 ビット乗算器の検証を数時間で行っている．検証時間は，通常，ビット長に依存して増加する（オーダは表現形式や表現の対象となる演算に依存する）．

1.2 本研究の概要

本稿では，ワードレベル検証およびビットレベル検証を，**プレスブルガー文**真偽決定手続き [5, 6] を用いて自動で行う手法を提案する．

プレスブルガー文（以下 P 文）とは，整数の集合 Z 上の変数, 定数，整数上の加減算+, −，大小比較=, <(など)，整数の定数倍，論理演算∧, ∨, ¬, ⊕(など)，および∀, ∃からなる閉論理式のことである[1]．

本手法の特徴は，幾つかの工夫を行ったプレスブルガー真偽判定手続きを用いることにより，各モジュールの演算ビット長 n が増えても，回路中の基本モジュールの数やその組合せ方が同じで，かつ仕様記述のサイズが n に依存していなければ，ワードレベル検証にかかる時間がほとんど増加しないことである（**5.1** で例を示す）．

以下，本手法の概略について述べる．

本手法では，各モジュール（および回路全体）のワードレベル仕様を，以下の (i),(ii) のいずれかのスタイルで記述する．

(i) **演算の定義を記述**：モジュールが行う演算の定義を，上述のプレスブルガー算術の演算を組み合わせて書く．例えば加算について，そのまま演算子+を使って `OUT = INa + INb` と書けばよい．この例では，仕様記述のサイズは，モジュールの演算ビット長に依存しない（記述サイズが演算ビット長に依存するような例は，**2.2.1** で示す）．

(ii) **関数シンボルを導入して記述**：プレスブルガー算術で直接扱えない演算などについては，それを表す関数シンボルを導入して書く．例えば 4bit 乗算について，`mult4(a,b)` という関数シンボルを導入し，`OUT = mult4(INa,INb)` を仕様とする．この例でも，仕様記述のサイズは，モジュールの演算ビット長に依存しない．

証明では，以下の検証条件（論理式）ϕが成立する事を示す[2]：

∀各変数 [(モジュール 1 の仕様 ∧ ···
∧モジュール n の仕様
∧モジュール間の接続関係) → 全体仕様]

(i) のスタイルで仕様を記述した場合，ϕが P 文になっているので，ϕ の真偽を直接（後述の）P 文真偽判定手続きで判定する．(ii) のスタイルで記述した場合，ϕが P 文でないので，「関数シンボルを自由解釈としてもϕが成り立つこと」を表す P 文ξを構成し，ξの真偽を P 文真偽判定ルーチンで判定する．その結果が真なら，ϕも成り立つと結論する（詳細は文献 [7] 参照）．(ii) の方法では，通常，導入した演算に関して数学的に成り立つ性質を検証者が導入し，ϕの前提条件部に（補題として）入れる．

証明には，高速化のための幾つかの工夫がなされた P 文真偽判定ルーチンを用いる．以下，本ルーチンの概略を説明する．

検証条件ϕ（またはξ）は，∀だけで束縛された冠頭標準形の P 文（P 文のサブクラス）になっている．それが真であることの証明は，「$\forall x\ P(x) \Leftrightarrow \neg\exists x \neg P(x)$」の関係を利用し，∃だけで束縛された冠頭標準形の P 文 $\exists x \neg P(x)$ が偽である事の判定に帰着して行う．∃だけで束縛された冠頭標準形の P 文は，**EPP 文**と呼ばれる．

本判定ルーチンでは，基本アルゴリズムとして，これまでに知られている最も速い EPP 文真偽判定法である

[1] 文献 [5, 6] 中での定義とは異なり，本稿では，整数型の変数だけでなく論理型の変数を持つ式も，P 文と呼ぶ．論理型の変数があっても，真偽は決定可能である．

[2] **2.1**，**2.3** で述べるが，モジュールや全体仕様間の整数変数は，ビット長や符合化の仕方が全く同じものどうしを対応づける．例えば，ある変数を（上位桁と下位桁をそれぞれ表す）二つの変数に分割する場合などは，その分割操作を行うモジュールを導入する．

Cooper・直井の判定法 [8] を用いる．その方法は，真偽判定するEPP文に対し，それと等価で一つ変数が少ないEPP文を構成する操作（**変数消去** quantifier elimination と呼ばれる）を繰り返し適用し，最終的に変数のない（定数だけの）式を得て真偽を判定するものである．

本判定ルーチンは，Cooper・直井の判定法に，変数消去順の制御や，変数消去中での（シンボリックな式変形操作による）式の簡単化・式の真偽評価などの処理を追加したものである．本ルーチンをワードレベル検証に用いた場合，変数消去順の制御により判定時間が相当に短縮され，実用時間での検証が可能となる．また，式の簡単化等を行うことにより，ほとんどの場合，真偽判定処理のうちモジュールの演算ビット長に依存して処理時間が変化する部分を，実行しなくて済む（詳細については **4.2.1** で述べる）．そのため，モジュールの組合せ方などが同じであれば，モジュールの演算ビット長が増えても，ワードレベル検証にかかる時間がほとんど増加しない．

実際に，本判定ルーチンにより，例えば n ビット加算器を二個用いて $2n$ ビット加算器を構成した場合のワードレベル検証を1分程度，n ビット加算器を三個用いて $3n$ ビット加算器を構成した場合の検証を65分程度，n ビット乗算器を四個用いて $2n$ ビット乗算器を構成した場合の検証を2分程度で行えた．証明にかかった時間は，演算ビット長 n が増えてもほとんど増加しなかった．

また，ビットレベル検証についても，演算ビット長が4ビット程度であれば，例えば加減算・論理演算を行うALU（74382）を6分程度と，実用的な時間で行えた．

以下，**2.** では，ワードレベルおよびビットレベルの仕様記述方法について，**3.** では，P文真偽判定手続きを用いた証明の方法について説明する．**4.** では，証明に用いるP文真偽判定ルーチンで用いる判定アルゴリズムと，その実装上の工夫について，**5.** では，検証実験の結果と考察について述べる．

2 回路の仕様記述法

2.1 階層的仕様記述

1.1 で述べたように，本手法では，回路の仕様記述を階層的に行う．図1に，8ビット加算器の仕様を，3段階に分けて階層的に記述した例を示す．以下，これをトップダウンに設計したものとして[3]，各段階（レベル1～3）の内容を説明する．

レベル1は，回路全体に対する要求仕様の記述である．仕様には，出力Yが入力A,B,CYINの和になることや，その和の値に応じてキャリーCYOUTやキャリー伝搬出力P,Gが正しく出ることなど，回路の入出力の間に成り立つべき関係が書かれている．ここで，入出力A,B,Y等はそれぞれ整数変数で表されており，その間の関係は，整数上の加算+，大小比較≤等の演算子を組み合わせてワードレベル仕様として書かれている[4]．

レベル2では，レベル1の8ビットアダーを，4ビット加算器を二個，Look-aheadキャリージェネレータを介して接続することにより，実現した．

レベル2の記述は，回路中の各モジュールの仕様と，それらの接続関係の記述からなる．ここで**モジュール**とは，加算器やバスの分割器（回路全体の入力A,Bから，各加算器の入力を作る）・統合器（各加算器の出力から，全体の出力Yを作る）など，何らかのデータ加工を行う部分回路のことである．各モジュールの仕様は，ワードレベル仕様として与えられる．

レベル3では，レベル2の各モジュールを，論理ゲートの組合せ等により実回路化した．**1.1** で述べたように，各モジュールの実回路の記述は，ビットレベル仕様と呼ばれる．ビットレベル仕様では，モジュールの入出力は全て論理型の変数で表され（図1中，例えばI[7..0]は，8個の論理型変数I7,…, I0の略記），回路の各出力が入力のどのような論理関数であるかが書かれている．

全体としての実回路は，レベル3の各モジュールを，レベル2で書かれた接続関係の通りに接続したものとなる．

2.2 回路全体および各モジュールの仕様の記述法

ここでは，モジュールのワードレベル仕様の記述法（回路の全体仕様としては，回路全体を一つのモジュールとみなし，そのワードレベル仕様を書く），およびビットレベル仕様の記述法について述べる．

また，記述・検証の対象とする回路に制約があるので，それについても述べる．

2.2.1 ワードレベル仕様の記述法

各モジュールのワードレベル仕様は，以下の三つの指定(a)～(c)からなる．

(a) 各入出力変数が，入力，出力のいずれであるかの指定. なお，本稿では，実回路中の全ての信号線は単方向で，二値（$true$,$false$）を取るものとする．双方向の信号線，トライステートの信号線，ワイヤードORなどを持つ回路については，記述・検証の対象としない．

(b) 各入出力変数のデータタイプ指定. モジュール M の入出力を表す変数を，整数型か論理型かにより，さらに整数型の場合，何ビットのビットベクトルをどのような方法で符号化したかにより，タイプ分けする（例えば，16ビットの符号なし整数，8ビットの2の補数，など）．仕様には，各変数ごと，そのデータタイプ名を指定する．

(c) 入出力変数間に成り立つ関係の指定. 以下に詳しく述べる． □

入出力変数間に成り立つ関係(c)は，以下の(i),(ii)のいずれかのスタイルで記述する．

(i) **演算の定義を記述：** 回路が行う演算の定義を，プレスブルガー算術の整数演算・論理演算を組み合わせて書く．

例えば加算については，そのまま演算子+を使って `OUT = INa + INb` と書けばよい．加算，大小比較，定数倍等の演算については，演算ビット長が変わっても，同じ形，同じサイズの式で入出力関係を書ける．

乗算など，プレスブルガー算術の整数演算を組み合わせるだけでは定義できない演算であっても，論理演算を併用する事で定義できる場合がある．例えば2bit乗算は，

```
OUT =      if in1=0 then 0
      else if in1=1 and in2
      else if in1=2 and 2・in2
      else              3・in2
```

というように，if文[5]を用い，各入力値に対する出力値のテーブルを書くことで定義できる．この場合，演算ビット長が増えると，テーブルは指数的に大きくなる．

(ii) **関数シンボルを導入して記述：** 異なる種類・演算ビット長の演算ごとに，それを表す関数シンボルを導入して書く（関数の定義を陽に書かない）．

例えば，4bit乗算については，その関数に対する `mult4(a,b)` という関数シンボルを導入し，式

[3] 実際には，要求仕様からトップダウンに，正しさを証明しつつ仕様記述・設計を進めてもよいし，実回路を設計してから，ボトムアップに中間レベルの仕様を構成しつつ証明を進めてもよい．

[4] ワードレベル仕様の入出力に，論理型の変数を用いてもよい．この例では，キャリー伝播出力P,Gは論理型の変数である．

[5] 「var = if $cond$ then $exp1$ else $exp2$」は，「$(cond \rightarrow var = exp1) \wedge (\neg\, cond \rightarrow var = exp2)$」の略記．

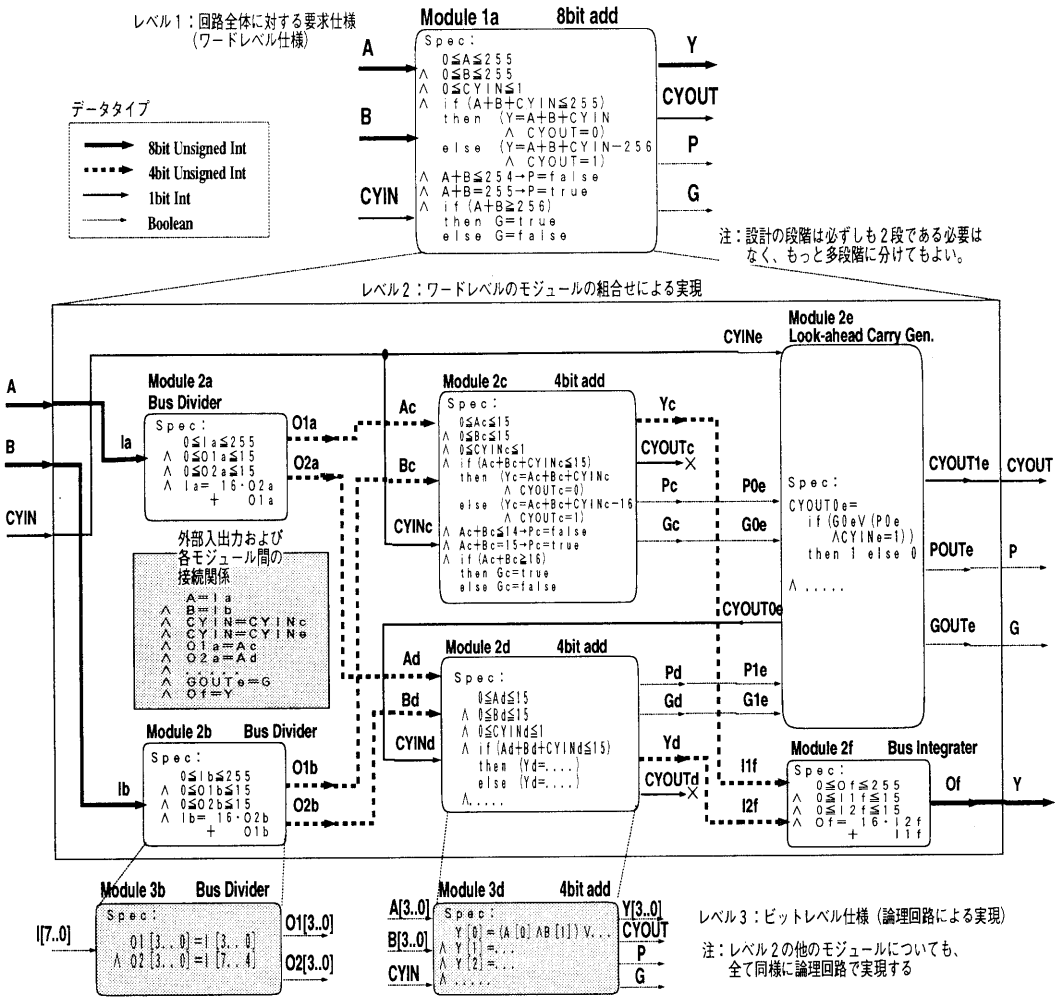

図 1: 8 ビットアダーの階層的仕様記述例（$2add_4$）

```
OUT = mult4(in1,in2)
```

を入出力関係の指定とする（図2参照）．あるいは，下位のモジュールの仕様記述で用いた関数シンボル（2bit 乗算 `mult2(a,b)`）を組み合わせることにより記述してもよい．

このスタイルでは，記述のサイズは演算ビット長に依存しない．

2.2.2 ビットレベル仕様の記述法

ビットレベル仕様の記述においても，**2.2.1** の (a)〜(c) を指定する（ただし変数のデータタイプは論理型のみ）．

モジュールの入出力を論理型の変数で表し，モジュール内の各論理ゲートの接続関係（結線）を，論理演算（∧,∨,⊕ など）を用いて記述する [9]．回路の入力から出力に至る途中のゲートの出力値も，それを表す論理変数を導入することで参照できる．

2.3 複数モジュールによる全体回路の実現時に指定する事項

ワードレベル仕様のモジュールを組み合わせて回路全体を実現するとき（図1のレベル2）は，以下の二つ (A),(B) を指定する．

(A) 各モジュールのワードレベル仕様.

2.2.1 で述べた．

(B) 各モジュールの接続関係.

接続する（端子を表す）変数どうしを，=で結ぶ．それらの等式の論理積をとって得られる論理式を，モジュールの接続関係の指定とする． □

なお，上記 (B) において，モジュール同士の接続は，以下の制約 1〜4 を満たすよう行わなければならない．

モジュール接続に関する制約 1：同じデータタイプ（**2.2.1** の (b) 参照）の変数どうしのみが，接続されていること．

モジュール接続に関する制約 2：入力変数は，必ず，いずれかの出力変数に接続されていること[6]．

モジュール接続に関する制約 3：出力変数どうしが接続されていないこと．

モジュール接続に関する制約 4：回路中に閉路（あるモジュール M の出力から，M の入力への，直接・間接のフィー

[6] どの入力にも接続されない出力は，存在してもよい．

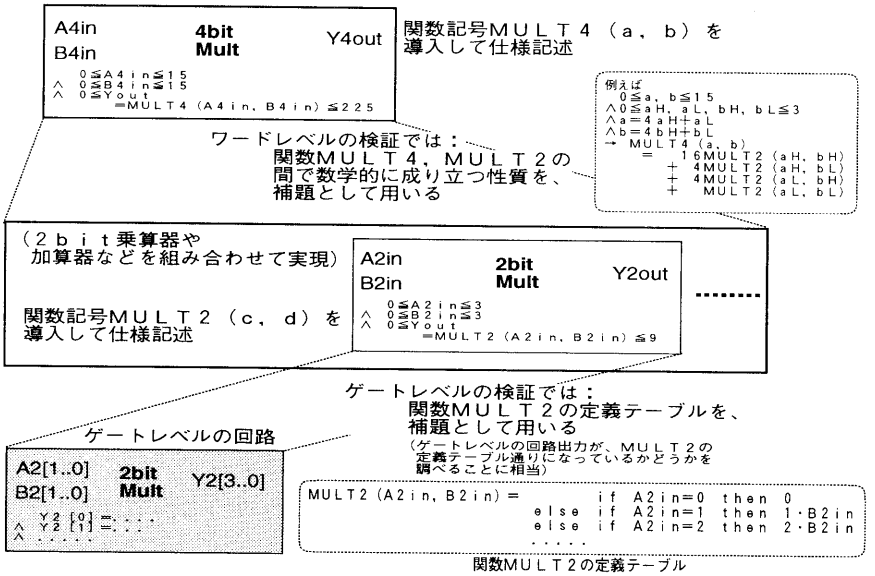

図 2: 関数シンボルを導入して仕様を記述した場合の検証方法（$4mult_4$）

ドバック）が存在しないこと． □

これらの制約が満たされているかどうかは，**3.** で述べる証明方法では通常チェックできないので，別途チェックしておく必要がある[7]．

3 プレスブルガー文真偽判定手続きを用いた検証法

1.1 で述べたように，ワードレベル仕様として書かれた幾つかのモジュールを接続したとき，そのもとで回路全体の仕様が正しく実現されていることの証明を，ワードレベル検証という．また，ビットレベルのモジュール（実回路）が，対応するワードレベル仕様を正しく実現していることの証明を，ビットレベル検証という．

図1では，レベル 2 の回路がレベル 1 の全体仕様を満たすことの証明がワードレベル検証，レベル 2 の各モジュールが対応するレベル 3 のモジュールで正しく実現されていることの証明がビットレベル検証にあたる．

いずれの検証でも，実現における任意の入力値 I_{impl}，および I_{impl} に対する（実現側の）出力値 O_{impl} について，$<I_{impl},O_{impl}>$ が仕様に書かれた入出力関係を満たすことを示す．

ここでは，ワードレベル検証およびビットレベル検証を，P 文真偽判定手続きを用いて行う方法について説明する．

3.1 ワードレベル検証の方法

ワードレベル検証では，以下の論理式（検証条件）ϕ が成り立つ事を示す：

∀各変数 [(補題（あれば）
∧ モジュール 1 の仕様 ∧ ···
∧ モジュール n の仕様
∧ モジュール間の接続関係) → 全体仕様]

ϕ が成り立つ事を示す方法は，各モジュールの仕様や全体仕様が，どのようなスタイルで書かれているかにより，以下のように異なってくる．

[7] 回路の入力から出力へ向かって，モジュールの接続関係をトレースしていく事により，機械的にチェックできる．

(1) 全てのモジュールの仕様および全体仕様が，プレスブルガー算術の演算だけを組み合わせて書かれている場合：
検証条件 ϕ は P 文になっているので，その真偽を，直接 P 文真偽判定手続きで判定する．その結果が真であれば，全体仕様は正しく実現されている．

もちろん，証明は完全に自動で行える．

(2) 関数シンボルを導入して仕様記述されたモジュールが（一つでも）ある場合：
検証条件 ϕ は P 文ではないので，その成立を直接 P 文真偽判定手続きで示すことはできない．

そこで，ϕ 中の関数シンボルを自由解釈としても ϕ が成り立つことを表す P 文 ξ を構成し（以下に述べる方法により，ϕ がどのような式であっても，必ず構成できる），ξ の真偽を P 文真偽判定ルーチンで調べる．その結果が真であれば，ϕ も成り立つと結論する．詳細は文献 [7] 参照．

ξ は以下のようにして構成する．プレスブルガー算術の演算子にはさまれた ϕ 中の全ての部分項を，それぞれ整数型または論理型の変数に置き換え（すでに変数である場合は除く），式の頭で ∀ で束縛する．ただし，文字列として同じ部分項は同じ変数に，異なる部分項は異なる変数に置き換える．例えば，ϕ を

$$\forall out \forall o1 \cdots [\ (o1 = mult2(i1H, i2H) \wedge \cdots) \rightarrow (out = mult4(in1, in2) \vee \cdots)\ ]$$

とすれば，関数シンボル $mult2(i1H, i2H)$,$mult4(in1, in2)$ をそれぞれ整数型の変数 $v1, v2$ に置き換えて ∀ で束縛した式

$$\forall out \forall o1 \forall v1 \forall v2\ [\ (o1 = v1 \wedge \cdots) \rightarrow (out = v2 \vee \cdots)\ ]$$

が ξ である． □

(2) の場合，通常，導入した演算に関して数学的に成り立つ性質を，検証者が補題（仮定）として検証条件 ϕ 中に入れないと，ξ が真にならない．例えば 4bit 乗算器を 2bit 乗算器の組み合わせで実現した場合，関数 $mult4$ と $mult2$ の間に成り立つ次のような関係を，補題として用いる（図 2参照）：

$$[\ 0 \le a, b \le 15 \wedge a = 4 \cdot aH + aL \wedge 0 \le aH, aL \le 3 \wedge b = 4 \cdot bH + bL \wedge 0 \le bH, bL \le 3$$

$$] \rightarrow mult4(a,b) = 16 \cdot mult2(aH,bH) + 4 \cdot mult2(aH,bL) + 4 \cdot mult2(aL,bH) + mult2(aL,bL)$$

3.2 ビットレベル検証の方法

ビットレベル検証では，以下の論理式ϕ'が成り立つ事を示す [9].

∀各変数 [(補題（あれば）
∧ ビットベクトルと整数値の対応
∧ ビットレベル仕様)
→ ワードレベル仕様]

ここで，**ビットベクトルと整数値の対応**とは，ビットレベル仕様M_bにおける論理型変数（通常，複数個）の値を，どのように符号化してワードレベル仕様M_wの整数変数の値とするかの指定である．例えば，M_bの二個の変数$b1, b0$を2ビット符合なし整数とみなして，M_wの整数変数iに対応付けるときは，

```
    b0 = if bit[0] then 1 else 0
and b1 = if bit[1] then 1 else 0
and i = 2 b1 + b0
```

がビットベクトルと整数値の対応の指定となる．2の補数表現などの場合についても，同様に指定できる．

ϕ'の成立を示す方法は，ワードレベルの検証の場合と同じである．ϕ'がP文になっている場合，P文真偽判定手続きによりその真偽を判定する．判定結果が真なら，ビットレベル仕様はワードレベル仕様を正しく実現している．ϕ'がP文でない場合（ワードレベル仕様の記述に，ユーザ定義の関数シンボルが使われている場合），補題として，その関数の定義テーブルを与える（図2参照）．

4 証明に用いるプレスブルガー文真偽判定ルーチンの概要

ワードレベル検証にかかる時間は，演算ビット長に依存しないか，依存しても少ないオーダ（例えばリニア）で抑えられることが望ましい．また，多くのモジュールを接続しても高速に検証できることが望ましい．

1.2で述べたように，**3.**の検証条件ϕが真であることの判定は，「$\forall x\ P(x) \Leftrightarrow \neg\exists x \neg P(x)$」の関係を利用し，EPP文が偽であることの判定に帰着して行う．

EPP文の真偽判定アルゴリズムは従来より幾つか提案されているが [6, 8, 10]，以下のいずれかの問題があり，ワードレベル検証に使うのは事実上不可能であった．

- 演算ビット長が多くなり，EPP文中の整数変数の係数値が大きくなると，判定時間が指数的に増加してしまう．
- モジュール数が多くなり，EPP文中の変数の数が多くなると，判定時間が指数的に増加してしまう．

  なお，従来の判定アルゴリズムでは，EPP文中の変数が数個程度であっても，判定に数時間以上もかかった（**5.**で用いる例題回路では，変数は10〜30個ほどある）．

本ルーチンは，これまでに知られている最も速いEPP文真偽判定法であるCooper・直井の判定法 [8] を，幾つかの新たな工夫を加えて実装したものである．本ルーチンでは，本稿で例題として用いた加算器等のワードレベル検証の場合，判定時間は数分程度（EPP文中の変数の数が20個程度，モジュール数にして6,7個程度の場合）であった．モジュールのビット数が増えても，全体の構造が同じなら，EPP文中の係数が大きくなるだけであり，それに対しては判定時間はほとんど影響を受けない．

以下，Cooper・直井の判定法の概要と，本ルーチンで行っている工夫について述べる．

4.1 本ルーチンで用いる真偽判定アルゴリズム

Cooper・直井の判定法では，真偽を判定するEPP文

$$\psi \triangleq \exists x_n \cdots \exists x_2 \exists x_1 F(x_1, x_2, \cdots, x_n)$$

中の各整数・論理変数$x_1, \cdots, x_n$ごと，ψ中の各不等式より定まる有限個の値で場合分け（変数消去と呼ばれる）を行う[8].

一つの整数変数x_iに対する場合分けでは，x_iに対応する新たな変数シンボルI_{xi}を導入する．そのI_{xi}については，λ_{xi}をϕより定まる正の整数定数として，取り得る範囲が$1 \leq I_{xi} \leq \lambda_{xi}$に定まっている（以下，$I_{xi}$を，変数$x_i$と区別し，レンジ属性付き変数と呼ぶ）．

全ての変数$x_1, \cdots, x_n$について場合分けを行うと，各場合ごとに，以下の一般形をした$I_{x1}, \cdots, I_{xn}$と定数だけからなるEPP文τが得られる．

$$\tau \triangleq \exists I_{x1} \cdots \exists I_{xn} \{ F'(I_{x1}, \cdots, I_{xn}) \wedge \rho_{x1} \mid \mu_{1,1} I_{x1} + \mu_{1,2} I_{x2} + \ldots + \mu_{1,n} I_{xn} + \nu_1 \wedge \cdots \wedge \rho_{xn} \mid \mu_{n,1} I_{x1} + \mu_{n,2} I_{x2} + \ldots + \mu_{n,n} I_{xn} + \nu_n \} \text{ where } 1 \leq I_{x1} \leq \lambda_{x1}, \cdots, 1 \leq I_{xn} \leq \lambda_{xn}$$

ここで，$c|exp$は，合同式「$exp \equiv 0 \pmod{c}$」(式expが整数定数cで割り切れること）を表す．また，各$\rho_{xi}, \mu_{i,j}, \nu_i$は整数定数である．

どの場合分けについても（その場合の）τが偽なら，ψは偽である．一つでもτが真の場合があれば，ψは真である．各τの真偽は，次のように調べる．まず，$F'(\cdots)$と∧で結合しているn個の合同式を，$I_{x1}, \cdots, I_{xn}$に関する連立1次合同式ζとみなし，各I_{xi}の範囲指定のもとで，ζの解（$<I_{x1}, \cdots, I_{xn}>$の組）を全て求める．連立合同式ζの解は，連立1次方程式における掃き出し法と類似した方法により求める．次に，求まった各解を$F'(\cdots)$に代入し，$F'(\cdots)$の真偽を調べる．どの解についても$F'(\cdots)$が偽ならτは偽，一つでも真ならτは真である．

4.2 判定アルゴリズム実装上の工夫

4.2.1 シンプレックス法による充足不能性チェック

連立合同式ζを解く直前に，τ中の$F'(I_{x1}, \cdots, I_{xn})$の充足不能性を，実数上の線形計画法（シンプレックス法）を用いてチェックする（実数上で充足不能なら，整数上でも充足不能．詳細は文献 [6] 参照）．充足不能であればζを解かず，すぐにτ全体を"$false$"とする．

この方法を使った事により，**5.**で示すワードレベル検証例では，モジュールの演算ビット長が多くなりEPP文の変数の係数値が大きくなっても，判定時間がほとんど増加していない．以下，その理由について簡単に述べる．

Cooper・直井の判定法において，変数の係数値に依存して処理時間が特に大きく変わる部分は，ζの解をF'に代入して真偽を調べる処理である．

5.で示すワードレベル検証例では，ほとんどの場合，$F'(\cdots)$が充足不能になる．そこで，本節で述べた方法を使うことにより，上記の処理をしなくても済む[9].

4.2.2 変数消去順の制御による場合分け回数の削減

EPP文については，どのような変数順で消去しても，正しく真偽判定を行える．変数消去順により，多くの場合，場合分けの回数が非常に大きく変わることが分かっている [11].

本ルーチンでは，場合分けを行う度に，以下のルールに従って次に消去する変数を決めていく．

---

[8] 論理変数については，trueの場合とfalseの場合に場合分けする．

[9] シンプレックス法による充足不能性チェックについては，変数の係数が大きくなっても処理時間は変わらない．

表 1: 加算器のワードレベル検証の結果
(Pentiam 200MHz 使用)

例題 (入力のビット長)	P 文中の変数の数	P 文中の係数の最大値	判定時間 (秒)	判定中の場合分け回数
$2add_2(4)$	13	4	54	3696
$2add_4(8)$	13	16	53	3696
$2add_8(16)$	13	256	53	3696
$3add_2(6)$	19	16	3945	155718
$3add_4(12)$	19	256	3954	155718
$3add_8(24)$	19	65536	3952	155718

表 2: 乗算器のワードレベル検証の結果
(Pentiam 200MHz 使用)

例題 (入力のビット長)	P 文中の変数の数	P 文中の係数の最大値	判定時間 (秒)	判定中の場合分け回数
$4mult_2(4)$	23	16	117	2630
$4mult_4(8)$	23	256	116	2570
$4mult_8(16)$	23	65536	112	2406

消去する変数を決めるルール 1 : 実現 (の各モジュール) 中の変数を, (証明する) 仕様中の変数よりも先に, 消去する変数として選ぶ. 実現中の変数については, 回路全体の入力側 (の変数) から出力側へ向かう順に, 消去する変数として選ぶ[10].

消去する変数を決めるルール 2 : ルール 1 で複数の候補が出た場合, 式の構文木の根からの深さが最も浅い位置に出現する不等式中の変数を選ぶ.

消去する変数を決めるルール 3 : ルール 1,2 で複数の候補が出た場合, それらのうち, 場合分けで代入する値の個数 ($\simeq$ その変数を含む (不) 等式の個数) が最も少ない変数を選ぶ. □

なお, ルール 1〜3 を適用しても複数の候補が出た場合は, その中から一つの変数をランダムに選ぶ.

4.2.3 式の簡単化による場合分け回数の削減

場合分けで変数に値を代入する度に, 式をなるべく簡単化する. それにより, 式中の変数の数が減り, 場合分け回数をかなり削減できることが多い.

Cooper・直井の判定法でも単純な簡単化処理は行うが, 本ルーチンでは, 以下のような, 変数が残った式に対する簡単化処理を追加してある.

(i) **冗長な不等式の削除 :** たとえば $cons_1 \geq cons_2$ であるとき, 部分式"$cons_1 < term \wedge term < cons_2$"を"$false$"に, 部分式"$cons_1 < term \wedge cons_2 < term$"を"$cons_2 < term$"に置換する等, 冗長な不等式の削除を行う.

(ii) **レンジ属性付き変数だけからなる不等式の真偽決定 :** EPP 文中の任意の不等式 $\theta \triangleq c_1 \cdot I_{x1} + c_2 \cdot I_{x2} + \cdots + c_n \cdot I_{xn} > d$ に対し, θの左辺の最大値 MAX と最小値 MIN を求める. MAX (あるいは MIN) は, c_iが正の I_{xi}に対してλ_{xi} (1), c_iが負の I_{xi}に対して 1 (λ_{xi}) を代入することで求まる. その結果, $MIN > d$ が真なら, θを"$true$"に置換する. 一方, $MAX > d$ が偽なら, θを"$false$"に置換する. いずれでもない場合は, θはそのままにしておく.

5 検証例

5.1 ワードレベル検証の実験結果

幾つかの例題回路のワードレベル検証を, **4.** で述べた真偽判定ルーチンを用いて行った. 回路の演算ビット数や構成の仕方 (どのような基本演算モジュールを幾つ組み合わせるか) を変えて, 加算器, 乗算器, および ALU の証明を行った結果を, それぞれ表 1, 表 2, 表 3 に示す[11]. なお, 設計誤りや補題の不足によって証明に失敗する事があるが, そのような場合, より短い時間で判定できることが多い.

加算器の検証実験では, n ビットの加算器を Look-ahead キャリージェネレータを介して m 個接続したとき, 全体として $m \cdot n$ ビットの加算器となることを証明した.

仕様記述では, 全体仕様, および実現中の各モジュールの仕様のいずれについても, 演算の定義 (**2.2.1** で述べたスタイル (i)) を記述した. 例題の名前「$x\ add_y$」は, y ビットの加算器を x 個接続した事を表す. 図 1の例が, 表 1の $2add_4$である.

証明は, 検証条件ϕの真偽を直接 **4.** の判定ルーチンで判定すること (**3.1** で述べた (1) の方法) により, 完全に自動で行えた. 補題は不要であった.

検証条件ϕは, yが異なっても x が同じであれば, 式中の整数変数の係数や整数定数の値が異なるだけで, 変数の数や, 式の構造などは全く同じである.

そのϕの真偽判定にかかった時間は, x が同じであれば, モジュールの演算ビット数 yに関わらず, ほとんど同じであった. EPP 文真偽判定中の場合分けの回数は, 全く同じである. これは, **4.2.1** で述べた判定ルーチンの工夫が, 有効に働いたからである. また, **4.2.2**, あるいは **4.2.3** の工夫をしない判定ルーチンでは, 表 1のいずれの例題でも, 判定に 1 日以上かかる (表 2, 表 3 中の例題でも同様).

一方, 回路のモジュール数が増加し, EPP 文中の変数の数が増えると, 判定時間は指数的に増加してしまう. このため, 実用時間内で一度に検証できるのは, モジュールが高々10 個程度 (P 文中の変数の数が 20〜30 個程度) の回路までであると思われる. それ以上のモジュールを含む回路の証明については, 証明の階層を深くし, 各段階の実現中のモジュール数を減らした上で, 証明をすべきと思われる.

乗算器の検証実験では, n ビット乗算器を四個, および $2n$ ビット加算器を三個用いて $2n$ ビット乗算器を構成した[12]とき, その実現が正しいことを証明した. 仕様記述では, 全体仕様, および実現中の各モジュールの仕様のいずれについても, ユーザ定義の関数シンボル `multN(a,b)`, および `mult2N(c,d)` を導入して記述した (**2.2.1** で述べたスタイル (ii)). 例題の名前「$4mult_y$」は, yビットの乗算器を四個接続した事を表す. 図 2の例が, 表 2の $4mult_2$ である.

証明は, **3.1** の (2) の方法により行った. 補題としては, **3.1** の (2) で例として示した関係式を導入した. ただし, 行う乗算のビット長により, 補題中に出現する各 `multN` の係数は変えてある. 検証は, 加算の場合と同様, モジュールの演算ビット長が増えても, ほとんど同じ時間で行えた.

ALU の検証実験では, 整数の n ビット加減算, および n ビットの論理演算を行う ALU (TTL の 74382) を m 個

[10] 仕様／実現の変数がどれか, 入力の変数がどれか等は, モジュールの接続関係や回路図などをトレースすることにより, 機械的に調べることができる.

[11] 真偽判定ルーチンにかける前処理として, 入出力 p_1,p_2,$\cdots$ が互いに接続されている ($p_1 = p_2 = \cdots$) とき, 検証条件ϕ中の p_2,$\cdots$を全て p_1に読み換える. これによって, 変数の数が減り, 真偽判定が速くなる. 各表の「P 文中の変数の数」には, この読み換えを行った後に残った変数の個数を示す.

[12] 二つの $2n$ ビット入力をそれぞれ上位・下位 n ビットずつに分け, それぞれ n ビット乗算器で掛けあわせて, 得られた sub-products を加算器で加えあわせた. このような実現の仕方では演算速度が遅いが, 証明は, n ビット乗算器の検証から始めて $2n$,$4n$,$\cdots$と階層的に行える. 今後, Wallace Tree などを用いて実現した場合の証明についても検討したい.

表 3: ALU のワードレベル検証の結果
（Pentiam 200MHz 使用）

例題（入力のビット長）	P 文中の変数の数	P 文中の係数の最大値	判定時間 (秒)	判定中の場合分け回数
$2alu_4(8)$	25	16	10	544
$2alu_8(16)$	25	256	10	544
$3alu_4(12)$	32	256	747	16311
$3alu_8(24)$	32	65536	750	16311

表 4: ビットレベル検証の結果
（Pentiam 200MHz 使用）

回路	P 文中の変数の数	判定時間（秒）	判定中の場合分け回数
2 ビット加算器 (TTL 7482)	17	0.5	97
4 ビット加算器 (TTL 74283)	37	123	2218
2 ビット乗算器 (Array Mult.)	16	0.5	80
4 ビット乗算器 (Array Mult.)	49	287	1478
4 ビット比較器 (TTL 7485)	36	130	6806
4 ビット ALU (TTL 74382)	49	372	12121

カスケード接続した時（Look-ahead キャリージェネレータは使っていない），全体として $m \cdot n$ ビットの ALU となることを証明した．仕様記述では，加減算については加算器の場合と同様に+（および−）を用い，論理演算については乗算器の場合と同様に $and4(a,b)$, $or4(a,b)$ などの関数を導入して，

```
OUT = if ALU モードが加算 then
          INa+INb
      else if ALU モードが AND 演算 then
          and4(INa,INb)
      else if .....
```

のように，if 文により各 ALU モードごとの出力を記述した．例題の名前「$x\ alu_y$」は，y ビットの ALU を x 個接続した事を表す．証明では，補題として，例えば「8 ビットのビットベクトル a,b の上位 4 ビットごとの論理積を α，下位 4 ビットごとの論理積を β としたとき，α を上位 4 ビット，β を下位 4 ビットとみなして得られる 8 ビットのビットベクトルは，a,b の論理積である」ことなどを用いた．ALU の場合も，モジュールの演算ビット長に関わらず，証明をほとんど同じ時間で行えた．

1.2 でも述べたように，上記のいずれの例題についても，各モジュールの仕様の記述サイズが演算ビット長 n によらない，という特徴がある．そのような場合，回路中のモジュールの数や組合せ方が同じ（例えば「$x\ add_y$」の x の値が同じ）であれば，各演算ビット長（y）が増えても，ワードレベル検証にかかる時間はほとんど増加しない．

5.2 ビットレベル検証の実験結果

4. で述べた判定ルーチンにより，2 ビット乗算器，4 ビット乗算器，TTL の 2 ビット加算器 7482，4 ビット加算器 74283，4 ビット比較器 7485，4 ビット ALU 74382 のビットレベル検証を行ってみた．

表 4に証明の結果を示す（表中，P 文の変数の数には，論理回路の中間ノードを表す変数も含まれる）．乗算器については，普通の multiplication array 方式で作成した論理回路を検証実験に用いたが，最適化などをした回路であっても，証明にかかる時間は同程度と思われる．

本手法によるビットレベル検証には，他手法と同様，モジュールの入力変数の数に対して，指数的な時間がかかる．本手法では，演算ビット長が 4 ビット程度であれば，実用的な時間で証明できる事が多い．

6 おわりに

本稿では，プレスブルガー文真偽判定ルーチンを用いたワードレベル検証・ビットレベル検証の方法について述べた[13]．本手法の特徴は，幾つかの工夫を行ったプレスブルガー真偽判定ルーチンを用いたことにより，各モジュールの演算ビット長 n が増えても，回路中のモジュールの数や組合せ方が同じで，かつ仕様記述のサイズが n に依存していなければ，ワードレベル検証にかかる時間がほとんど増加しないことである．その証明は，本稿で例題として用いた加算器や乗算器程度であれば，十分実用的な時間で自動で行える．

今後の課題としては，本手法により，大規模なベンチマーク回路の証明をどの程度の時間で行えるかを調べることや，整数以外のデータタイプの演算（浮動小数点演算など）を行う回路の証明法について検討すること等が挙げられる．

参考文献

[1] Randal E. Bryant and Yiring-An Chen: "Verification of Arithmetic Circuits with Binary Moment Diagrams", Proc. 32nd Design Automation Conference (DAC 95), pp.535−541, (1995).

[2] A.J.Camilleri, M.J.C.Gordon and T.F.Melham, "Hardware verification using higher-order logic", HDL Descriptions to Guaranteed Correct Circuit Designs, D.Borrione(editor), pp.43−67, N.Holland, Amsterdam (1987).

[3] S.Owre, J.M.Rushby, N.Shankar and M.K.Srivas : "A Tutorial on Using PVS for Hardware Verification", LNCS Vol. 901 (TPCD94), pp.258−279, Springer Verlag (1995).

[4] Deepak Kapur, M. Subramaniam: "Mechanically Verifying a Family of Multiplier Circuits", LNCS Vol. 1102 (CAV96), pp.135−146, Springer Verlag (1996).

[5] M.Presburger: "Uber die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen", in welchen die Addition als einzige Operation hervortritt, in Comptes-Rendus du ler Congress des Mathematiciens des Pays Slavs (1929).

[6] 東野輝夫, 北道淳司, 谷口健一: "整数上の線形制約の処理と応用", コンピュータソフトウェア, Vol.9, No.6, pp.31−39 (1992).

[7] Kitamichi,J., Morioka,S., Higashino,T. and Taniguchi,K.: "Automatic Correctness Proof of the Implementation of Synchronous Sequential Circuits using an Algebraic Approach", LNCS Vol. 901 (TPCD94), pp.165-184, Springer Verlag (1995).

[8] 直井邦彰, 高橋直久: "プレスブルガー算術を用いた Infeasible Path 検出の高速化技法", 信学技報, SS95−19, pp.71−78 (1995).

[9] 森岡澄夫, 北道淳司, 東野輝夫, 谷口健一: "整数上の論理式の恒真性判定アルゴリズムを用いた組合せ論理回路の実現の正しさの証明", 第 49 回情処全大 4L−01 (1994).

[10] Cooper,D.C.: "Theorem Proving in Arithmetic without Multiplication", *Machine Intelligence*, No.7, pp.91-99(1972).

[11] 森岡澄夫, 東野輝夫, 谷口健一: "全ての変数が存在記号で束縛された冠頭標準形プレスブルガー文の真偽判定プログラム", 信学技報, SS95−18, pp.63−70 (1995).

[13] 本稿で用いたプレスブルガー文判定ルーチンのバイナリ，および検証実験に用いた例題は，
`http://sunfish.ics.es.osaka-u.ac.jp/program.html`
で公開している．