ボトムアップ定理証明器の効率的アルゴリズムとその評価

長谷川 隆三[†] 越村 三幸[†] 藤田 博[††]

本論文では遅延モデル生成という考えに基づき，効率の良いボトムアップ定理証明器を実現するためのアルゴリズムを提案する．モデル生成法に基づく定理証明の仕事は節集合を充足させるようなアトム集合（モデル候補）の生成とそのテストであるが，一般に生成されるアトム集合が組合せ的に爆発し，計算量と必要なメモリ空間が膨大になる傾向がある．これを解決するには，証明過程をgenerate-and-testと捉え，生成プロセスをテストプロセスからの要求によって制御することが重要である．提案するアルゴリズムはこの考えを実現したものであり，証明に寄与しない不必要なアトムの生成を抑制する．本アルゴリズムを採用することにより，計算量およびメモリ量のオーダを大幅に（節の前件リテラル数をnとすると，n乗根分）低減できることを解析的に示し，さらに実験によってその効果を定量的に評価する．

New Algorithms to Implement Efficient Bottom-up Provers and Their Evaluation

RYUZO HASEGAWA, [†] MIYUKI KOSHIMURA[†] and HIROSHI FUJITA[††]

This paper presents several algorithms for implementing efficient bottom-up theorem provers based on lazy model generation. The tasks of the model generation based prover are the generation and testing of atoms which are to be the elements of a model for the given theorem. A problem with this method is explosion in the number of generated atoms and in the large computation amount and memory space required for proof process. To solve this, it is important to view proving processes as *generate-and-test* processes and to control the generation process by a demand from the testing process. The algorithms given in this paper, based on the above idea, avoids generating unnecessary atoms that are irrelevant to obtaining proofs. In this paper, we give an analysis to show that with these algorithms, the time and space complexity can be decreased by several orders of magnitude (for a clause with n antecedent literals, by a factor of nth root). We also confirm the practical effects through some experiments.

1. はじめに

本研究では並列論理型言語KL1[1]の特長を活用することにより，高速な一階述語論理定理証明器の開発を目指している．我々はKL1ベースの定理証明器の開発にあたり，SATCHMO[2]のモデル生成法を採用した．その理由は以下のとおりである．1）モデル生成法の場合，広範囲な有限領域の問題に対して単一化が不要であり照合ですむので，KL1の高速なガード単一化（パタン照合機能）を活用できる．2）数学的定理のように深い推論（長い証明）を必要とする問題を解くにあたって，モデル生成法や超導出（hyper-resolution）に基づくボトムアップ定理証明器は，探索空間を狭めるための各種の技法，すなわち，補題化，包摂テスト，削除戦略などを容易に組み込むことができる．

しかし，効率の良いボトムアップ定理証明器を実現するには，前向き推論の過程で証明に寄与しない無駄なアトム集合が生成されるのを極力抑え，全体の計算量および必要メモリ量を削減することが重要な課題となる．特にŁukasiewiczの論理[3]や類似のcondensed detachment問題[4]等を証明する場合は，一般に推論段数が長大になる傾向があり，このアトム過剰生成問題はより一層深刻になる．

従来型定理証明器の代表格であるOTTER[5]は，種々の証明手続きや不要アトムの削除戦略を組み込むことができるように汎用的かつ柔軟に設計されている．しかしながら，節を生成用とテスト用に明確に区別して実装していないので，超導出のような前向き推論を行った場合，生成されるアトム集合の組合せ的爆

† (財)新世代コンピュータ技術開発機構
Institute of New Generation Computer Technology (ICOT)
†† 三菱電機(株)中央研究所
Central Research Laboratory, Mitsubishi Electric Corporation

発を引き起こす危険がある．またgenerate-and-testという考えのないナィーブな実装では，アトムの過剰生成によって証明過程中に無駄な計算が生じる可能性があり，さらには，メモリオーバフローを引き起こす恐れもある．

一方，Prologベースの定理証明器であるSATCHMOの場合，節集合を非ホーン節集合とホーン節集合に分離し，前者には前向き推論を適用してモデル拡張（新アトムの生成とホーン節集合への追加）を行い，後者にはPrologの後向き推論を直接利用してモデル棄却テスト（反駁）を試みる．このモデル拡張と棄却テストはassert/retractおよびPrologの後戻り機構を用いて，逐次的に完全に同期して行われているため，生成プロセスが暴走しテスト待ちのアトムを過剰生成する危険は生じない．しかしながら，アトムを一つ生成するたびに棄却テストを行うという固定化された制御の中に戦略を採り入れるのは難しい．例えば，生成されたアトムに重み付けを行い，最小のアトムを優先的に選んでモデル候補の拡張を行うというソーティング戦略は使えない．

上述の両システムの問題点を克服するためには，前向き推論の証明過程をgenerate-and-testと捉え，生成プロセスとテストプロセスを要求駆動的に制御する遅延モデル生成（Lazy Model Generation）[6]の考えが重要である．

本論文では，この考えに添って，ボトムアップ定理証明器の効率改善のための2種のアルゴリズム，全テストおよび遅延アルゴリズムを提示する．前者は手続き指向のアルゴリズムであり，OTTERと同様に生成プロセスはアトム集合を一度に生成する．しかし，その要素に対するテストがすべて完了するまでは次の生成を行わない点が異なる．一方，後者はプロセス指向のアルゴリズムであり，テストプロセスから要求された時のみ生成プロセスが指定個数分のアトムを生成するという要求駆動の考え方に基づいている．

さらに，前件リテラルが1個の単位負節に対する最適化技法を示し，計算量とメモリ量の観点から上記のアルゴリズムおよび最適化の効果を解析的に評価する．最後に，condensed detachment問題を対象にその有効性を定量的に検証する．以下では，ボトムアップ定理証明器の基準としてモデル生成法を用いる．

2. モデル生成法

モデル生成法は与えられた節集合のモデル（節集合を充足するアトムの集合）を，前向き推論によって機械的に求める証明手法である．本文を通じて，節$\neg A_1 \vee \cdots \vee \neg A_n \vee C_1 \vee \cdots \vee C_m$は以下のような含意形式で記述する．

$$A_1, \cdots, A_n \rightarrow C_1 ; \cdots ; C_m$$

ここで，$A_i (1 \leq i \leq n ; n \geq 0)$および$C_j (1 \leq j \leq m ; m \geq 0)$はアトムである．'→'の左辺（のアトム）を前件（リテラル），右辺（のアトム）を後件（リテラル）という．前件は$A_1, \cdots, A_n$の連言（','），後件は$C_1, \cdots, C_m$の選言（';'）である．節の前件部が空（$n=0$）の場合$A_0 = true$と置いて正節（positive clause），後件部が空（$m=0$）の場合$C_0 = false$と置いて負節（negative clause），それ以外（$n \neq 0, m \neq 0$）の場合は混合節（mixed clause）と呼ぶ．*true*および*false*は，それぞれ真，偽を表す特別のアトムである．さらに，混合節をgenerator節，負節をtester節とも呼ぶ．

モデル生成法には次の二つの規則がある．規則中Mは構成途中のモデルを表し，これをモデル候補と呼ぶ．

- モデル拡張規則：ある代入σのもとで，ある混合節もしくは正節の各前件リテラル$A_i\sigma$があるモデル候補Mで充足されており，かついずれの後件リテラル$C_j\sigma$もMで充足されていないとき，$C_j\sigma$をMに加えてMを拡張する．
- モデル棄却規則：ある代入σのもとで，ある負節の各前件リテラル$A_i\sigma$があるモデル候補Mで充足されるとき，Mを棄却する．

ここで，$(A_1, \cdots, A_n)\sigma$を得る過程を，節の前件リテラルとモデル候補アトムとの連言照合（conjunctive matching, CJM）と呼ぶ．正節の前件（*true*）は任意のモデルで充足可能であることに注意されたい．

モデル生成法の証明では，まず初期モデル候補集合として$\mathcal{M} = \{\emptyset\}$から始め，上記のいずれの規則も適用することができなくなった時点で，最終的なモデル候補の集合$\mathcal{M}$を出力する．このとき，$M \in \mathcal{M}$は与えられた節集合のすべての節を充足しているのでモデルとなる．もし，$\mathcal{M} = \emptyset$であれば，すべてのモデル候補が棄却されてモデルが存在しないため，その節集合は充足不能である．

3. モデル生成アルゴリズム

本章では，まずモデル生成法の基本的な実現アルゴリズムを説明し，これに対する2種の改良アルゴリズムを示す．そして，単位負節による棄却テスト（単位反駁）に関するさらなる最適化技法を示す．以下の節では，議論を簡単にするため問題はホーン節のみで与えられるものと仮定するが，ここでの考え方は非ホーン節で記述された問題にも一般化可能である．

3.1 基本アルゴリズム

図1に示す基本アルゴリズムは，モデル生成法を幅優先探索で実行するアルゴリズムであり，OTTERで実現されている超導出と本質的に同一のアルゴリズムである*．ここで，Mはモデル候補を，Dはモデル拡張候補(モデル拡張規則の適用の結果としてMに付加されるべきアトムの集合)を，ΔはDの部分集合をそれぞれ表す．MとDの初期値は空集合および正節の後件部のアトムの集合である．

このアルゴリズムのサイクルを一回りする間に，(1) Dの部分集合Δを選び**，(2) ΔとMを使って棄却テスト(tester節に対する連言照合)を行う．(3) 反駁に成功するとアルゴリズムは停止する．さもなければ，(4) ΔとMを使ってモデル拡張(generator節に対する連言照合)を行い，(5) 新しく生成されたアトム集合*new*の$M \cup D$に対する包摂テストを行う．このサイクルの始めにDが空ならば反駁が失敗，すなわちモデルが見つかったことになり，このアルゴリズムは停止する．

連言照合および包摂テストはそれぞれ以下のような集合上の関数として表される．

$$CJM_S(\Delta, M) = \{C\sigma \mid A_1, \cdots, A_n \rightarrow C \in S \wedge \forall i (1 \leq i \leq n) A_i\sigma \in (M \cup \Delta)\sigma \wedge \exists i (1 \leq i \leq n) A_i\sigma \in \Delta\sigma\}$$

$$subsTest(\Delta, M) = \{C \in \Delta \mid \forall B (B \in M) \not\exists \sigma B\sigma = C\}$$

ここで，CJM_Sは節集合Sに対する連言照合関数を意味し，連言照合対象がtester節集合TSのときCJM_{TS}, generator節集合GSのときはCJM_{GS}と表記する．ただし，$M \cup \Delta$に対して節$A_1, \cdots, A_n \rightarrow C$の連言照合を行う場合，$M \cup \Delta$の要素の$n$個の組すべてについて$A_i (i=1, \cdots, n)$との照合を行う必要はなく，$n$個のうち少なくとも一つは$\Delta$の要素である組についてのみ照合を行えばよい[7]．上記の連言照合関数はこのように冗長性を取り除いた操作を表している．また，tester節に対する連言照合が成功したとき，CJM_{TS}は$\{false\}$を返すものとする．

```
M := ∅;
D := {A | (true → A) ∈ a set of given clauses};
while D ≠ ∅ do begin
    select(Δ, D); D := D − Δ;                (1)
    if CJM_TS(Δ, M) ∋ false                  (2)
        then return(unsat);                  (3)
    new := CJM_GS(Δ, M);                     (4)
    M := M ∪ Δ;
    new' := subsTest(new, M ∪ D);            (5)
    D := D ∪ new';
end return(sat)
```

図1 基本アルゴリズム
Fig. 1 Basic algorithm.

基本アルゴリズムの動作を把握するため，次の節集合を考えよう．

C_1: $true \rightarrow p(a)$.
C_2: $p(X), p(Y) \rightarrow p(f(X, Y))$.
C_3: $p(f(f(a, a), f(a, a))) \rightarrow false$.

まず，MとDの初期値として，$\langle M_0, D_0 \rangle = \langle \emptyset, \{p(a)\} \rangle$が設定される．簡単のため，$D$からの$\Delta$の選択法は生成順とする．1サイクル目では，$D_0$から$\Delta_1 = \{p(a)\}$を選び，$C_3$を用いて棄却テストを行う．反駁に失敗する($CJM_{\{C_3\}}(\Delta_1, M_0) \not\ni false$)ので，$C_2$を用いてモデル拡張を行い，$new_1 = CJM_{\{C_2\}}(\Delta_1, M_0) = \{p(f(a, a))\}$を生成する．2サイクル目では，$\langle M_1, D_1 \rangle = \langle \{p(a)\}, \{p(f(a, a))\} \rangle$となる．同様に，$D_1$から$\Delta_2 = \{p(f(a, a))\}$を選び，$C_3$により棄却テストが行われた後，$C_2$により$new_2 = \{p(f(a, f(a, a))), p(f(f(a, a), a)), p(f(f(a, a), f(a, a)))\}$が生成される．この後，$new_2$から三つのアトムが順次選ばれ，5サイクル目で$\Delta_5 = \{p(f(f(a, a), f(a, a)))\}$に対する棄却テストを行うことにより反駁に成功し($CJM_{\{C_3\}}(\Delta_5, M_4) \ni false$), *unsat*が返される．

3.2 全テストアルゴリズム

図2に基本アルゴリズムを改良した全テストアルゴリズムを示す．このアルゴリズムでは，(1) DからΔ

```
M := ∅;
D := {A | (true → A) ∈ a set of given clauses};
while D ≠ ∅ do begin
    select(Δ, D); D := D − Δ;                (1)
    new := CJM_GS(Δ, M);                     (2)
    M := M ∪ Δ;
    new' := subsTest(new, M ∪ D);            (3)
    if CJM_TS(new', M ∪ D) ∋ false           (4)
        then return(unsat);
    D := D ∪ new';
end return(sat)
```

図2 全テストアルゴリズム
Fig. 2 Full-test algorithm.

* OTTERでは，次の小節で述べる全テストアルゴリズムにあるように，*new*に含まれる各アトムに対し，生成後すぐに単位反駁を行うという改良が施されている．

** DからのΔの選択法には，アトムの生成順に取り出す方法や重みが最小のアトムを優先する方法などがある．

を選び，（2）Δと M を使ってモデル拡張を行い，Δに対する次の世代のアトム集合 *new* を生成する．（3）$M \cup D$ に対して *new* の包摂テストを行った後，（4）包摂テストを通過した *new'* と $M \cup D$ を使って棄却テストを行う．

この修正は一見些細なようであるが，次章で述べるように，これによって全計算量とメモリ量が大幅に削減される．本アルゴリズムの重要な点は以下のとおりである．Δのかわりに新しく生成された *new* のすべてのアトムに対して包摂テストを施した後，棄却テストを行う．この結果，*new* の中に棄却アトム*(これを X とする)があった場合，直ちに偽が検出される．基本アルゴリズムとは異なり，棄却アトムが見過ごされ，D に付加されてしまうことはない．したがって，棄却アトムが生成された後に余計なアトムが生成/テストされることはない．前節3.1の例では，2サイクル目で new_2 が生成された段階で，new_2 の各アトムに対し棄却テストが行われ，偽が検出される．

3.3 遅延アルゴリズム

図3に基本アルゴリズムの別の改良である遅延アルゴリズムを示す．このアルゴリズムでは，generator 節に対応するプロセスと tester 節に対応するプロセスが交信しながら独立並行に動作するものと考える．

```
process tester:
  repeat forever
    request(generator, Δ);                    (1)
    Δ' := subsTest(Δ, M ∪ D);                 (2)
    if CJM_TS(Δ', M ∪ D) ∋ false              (3)
      then return(unsat);
    D := D ∪ Δ'.

process generator:
  repeat
    while Buf = ∅ do begin                    (1)
      select({e}, D); D := D − {e};           (2)
      Buf := delayCJM_GS({e}, M);             (3)
      M := M ∪ {e} end;
    wait(tester);                             (4)
    Δ :=forceBuf;                             (5)
  until D = ∅ and Buf = ∅; return(sat)
```

図3 遅延アルゴリズム
Fig. 3 Lazy algorithm.

* 棄却アトム(falsifying atom)とは，X 自身のみで負節の前件部を直接満たすか，または X と $M \cup D$ 中のアトムとの組で負節の前件部を満たすようなアトム X のことである．

tester プロセスは，（1）generator プロセスにΔを要求し，（2）これまでのテスト済みのアトム集合 $M \cup D$ に対してΔの包摂テストを行った後，（3）Δと $M \cup D$ を用いて棄却テストを行う．

generator プロセスは，（1）1回のモデル拡張の結果を蓄える *Buf* が空のとき，これを埋める準備をするため，（2）D からアトム e を選んで，（3）*Buf* にモデル拡張コード(continuation)を設定しておく(**delay***CJM*)．そして，（4）tester プロセスからのΔの要求を待って，（5）*Buf* を起動し(**force** *Buf*)，Δを送出する．

ここで，**delay** はオペレータで，被作用子の関数の実行を遅延する．したがって，（1）～（3）の時点で $CJM_{GS}(\{e\}, M)$ は直ちには計算されず，*Buf* 上にコードとして置かれる．その後，（5）で **force** オペレータが *Buf* に作用する時点で，**delay** された関数が起動され，必要とされるだけのアトムが作られる．これにより，毎回 tester プロセスが要求するΔ分のアトムだけが実際に生成されるわけである．なお，*Buf* には残りの生成を継続するために再び **delay** された *CJM* 関数コードが置かれる．

M，D へのアトムの登録は，アトムの生成順序およびこれらに対するテストの順番が基本アルゴリズムと同じになるように行われる．したがって，遅延アルゴリズムは基本アルゴリズムにおけるアトム間の論理的な生成順序を保ちつつ，生成およびテストの相対的な速度を均等化したものといえる．これにより，アトムの過剰生成による計算およびメモリ消費の無駄をなくすことができるわけである．

3.4 単位負節に対する最適化

単位負節 $A \rightarrow false$ に対しては，上述の三つのアルゴリズムはさらに改良できる．これには二通りの方法がある．

一つは先読み方式と呼ばれる動的な方法である．これは，モデル拡張の過程において，単位負節による棄却テストを施すためだけに，アトムを先回りして生成するものである．すなわち，*new* を生成した直後に $(new := CJM_{GS}(\Delta, M))$，次の段階で生成されるべき new_{next} も生成する $(new_{next} := CJM_{GS}(new, M \cup D))$．そして new_{next} に対して単位負節による棄却テストを行う．このテストに失敗したとき，*new* はこれまでどおり記憶するが，new_{next} は廃棄する．

new_{next} を記憶しないで良い理由は，単位負節による棄却テストが M も D も必要とせず，new_{next} 自身で行えるからである．しかし2個以上のリテラルを持つ負節の場合は，new_{next} を廃棄してしまうと new_{next} ど

うしの組合せに対する棄却テストができず，棄却テストが不完全になる．new_{next} は次の段階の new として再び生成されてくるため，同一のアトムの組合せに対して連言照合が再度行われる場合が生じるが，これによる計算量の増加は全計算量に比べてオーダ的に無視できる．詳細は次章で議論する．

もう一つの方式は，部分評価を使った静的な方法である．すなわち，単位負節と generator 節の後件部を単一化し，部分評価された非単位負節を得る．

Generator : $A_1, A_2 \rightarrow C$.

Unit tester : $A \rightarrow false$.

⇩

Non-unit tester : $A_1\sigma, A_2\sigma \rightarrow false$.

where $C\sigma = A\sigma$

部分評価方式でも先読み方式と同様に連言照合における重複が存在する．しかし，部分評価方式では証明器そのものが修正されることはないので，先読み方式より単純といえる．さらに負節中の束縛情報を generator 節に伝播できるため，探索空間の刈り込みが可能になる場合もある．

以上述べた最適化を施すことによって，負節が単位節のときには，テスト可能な空間が広がるため，全計算量と必要メモリ量の改善が期待できる．

4. アルゴリズムの複雑さの解析

本章では，前章で提示された各アルゴリズムについて，計算量とメモリ量の観点からアルゴリズムの複雑さを議論する．

議論をできるだけ簡単にするために次のことを仮定する．1) 与えられる問題は前件部が2個，後件部が1個のリテラルのみからなる generator 節1本と，高々2個のリテラルからなる tester 節1本のみを含む．2) D から取り出される Δ は単一のアトムからなると仮定する．3) 連言照合で単一化に成功した結果生成されるアトムが包摂テストを通過する割合(生存率と呼ぶ)を $\rho(0 \leq \rho \leq 1)$ とする．また，4) Δ の取り出し順序およびモデル拡張におけるアトムの生成順は，アルゴリズムを問わず一定であるとする．

アルゴリズムの複雑さの解析にあたって，**図4**に示すように，生成されるアトム $a_1, a_2, \cdots$ に生成順に番号付けを行い，これをマトリックス上に並べる．このマトリックスは2個の前件リテラル L_1, L_2 に対する連言照合操作を表しており，縦軸は第一前件リテラル L_1 に，横軸は第二前件リテラル L_2 に照合させるモデル候補アトムの番号である．i 行 j 列の要素は，L_1 に i 番目のアトム a_i，L_2 に j 番目のアトム a_j を照合させた結果生成されたアトムの番号を表す．以下では，前件リテラル対 $\langle L_1, L_2 \rangle$ とアトム対 $\langle a_i, a_j \rangle$ の照合操作を $i \times j$ と表す．

×	1	2	3	...	i
1	2	4	8		1×i
2	3	5	9	...	2×i
3	6	7	10		3×i
:		:			:
i	i×1	i×2	i×3	...	i×i

図4 連言照合操作
Fig. 4 Conjunctive matching operation.

図4は，D の初期値として1個のアトム a_1 が与えられたとき，その番号を1とし，1×1の結果を2，2×1の結果を3，1×2の結果を4，2×2の結果を5，というように生成アトムに番号をふることを示している．

ここで，i 番目の(モデル候補)要素 a_i に対するモデル拡張 $CJM_{GS}(\{a_i\}, M)$ とは，M の全要素 $a_1, \cdots, a_{i-1}$ に対して $i \times (1 \cdots i-1)$, $(1 \cdots i-1) \times i$, $i \times i$ の連言照合を行い，新たなモデル拡張候補を生成することを指す．i 番目の要素に対するモデル拡張 $CJM_{GS}(\{a_i\}, M)$ が完了した時点では，総計 i^2 回の連言照合が行われ，生存率を ρ とすると，総計 $\lceil \rho i^2 \rceil$ 個のアトムが $M \cup D$ に格納されている*．

4.1 基本アルゴリズム

図5は基本アルゴリズムにおける計算量と必要なメモリ空間量を図示したものである．左の正方形の面積は generator 節で行われる連言照合の回数を表す．右の正方形の面積は2個のリテラルを持つ負節で行われる連言照合の回数を表し，この正方形の下の線の長さは単位負節の場合の連言照合の回数を表す．M, D は，それぞれの領域で行われる連言照合の結果生成されるアトム集合である．基本アルゴリズムでは，M の要素に対してはモデル拡張および棄却テストが完了し，D の要素に対してはモデル拡張も棄却テストもまだ行われていない．

図5は，*false* すなわち偽が検出された時点のものである(以後，図では *false* を ⊥ と表示)．ここで，m 番目の要素 a_m に対するモデル拡張 $CJM_{GS}(\{a_m\}, M)$ によって，棄却アトム X が生成されるものとする．このアトムは，$\lceil \rho(m-1)^2 \rceil + 1$ 番目から $\lceil \rho m^2 \rceil$ 番目のいずれかの要素であるが，簡単のため，$\lceil \rho m^2 \rceil$ 番目とする．

* i 個のアトムが M に，$\lceil \rho i^2 \rceil - i$ 個のアトムが D に格納される．

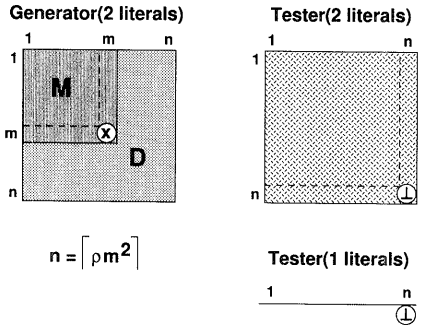

図5 基本アルゴリズムの複雑度
Fig. 5 Complexity of basic algorithm.

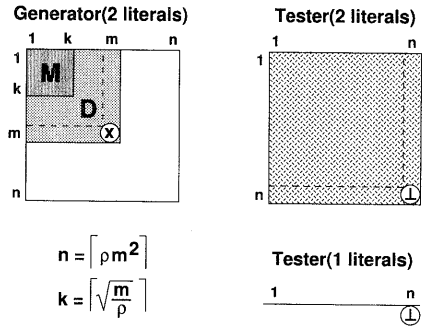

図6 全テスト/遅延アルゴリズムの複雑度
Fig. 6 Complexity of full-test/lazy algorithm.

基本アルゴリズムでは，まずモデル拡張候補 D から要素 Δ を一つ取り出し，tester 節による棄却テスト $CJM_{TS}(\Delta, M)$ を行う．全テストアルゴリズムとは異なり，Δ に対するモデル拡張 $CJM_{GS}(\Delta, M)$ の結果である *new* に対して棄却テストを行っていないため，棄却アトム X が *new* に含まれていてもこの時点では見過ごされてしまい，一旦 D に格納されることになる．したがって，D から X が取り出されて棄却テストが行われるまで，偽であることは判明しない．

この結果，$m+1$ 番目から $\lceil \rho m^2 \rceil$ 番目の要素に対する無駄な連言照合が generator 節で行われてしまい，X に対する棄却テストが終了した時点では，generator 節で $\lceil \rho^2 m^4 \rceil$ 回の連言照合が行われることになる．一方 tester 節では，2 リテラルの場合は $\lceil \rho^2 m^4 \rceil$ 回，1 リテラルの場合は $\lceil \rho m^2 \rceil$ 回の連言照合が行われる．そして $M \cup D$ の中には，$\lceil \rho^3 m^4 \rceil$ 個のアトムが保持される．したがって，基本アルゴリズムの計算量およびメモリ量は $O(m^4)$ である．

4.2 全テストアルゴリズム

図6は，全テストアルゴリズムにおいて *false* が検出された時点の図である．X, M は基本アルゴリズムの時と同じ意味であるが，D については，その要素に対してモデル拡張(generator の連言照合)は開始されていないのに，棄却テストは完了しているという点で異なる．

全テストアルゴリズムでは，棄却テストは Δ に対してではなく，Δ に対するモデル拡張の結果である *new* に対して行われている．したがって，m 番目の要素 a_m に対するモデル拡張 $CJM_{GS}(\{a_m\}, M)$ の結果として棄却アトム X が生成されると，このアトムに対する棄却テストによって直ちに偽であることが判明する．このとき，2 リテラルの tester 節で行われる連言照合の回数は，$M \cup D$ の個数が $\lceil \rho(m \times m) \rceil$ なので*，$(\rho(m \times m))^2 = \rho^2 m^4$ となり，これは基本アルゴリズムと同じである．

一方，generator 節に関しては，1 から m 番目の要素に対してのみ連言照合が行われ，$m+1$ 番目から $\lceil \rho m^2 \rceil$ 番目の要素** に対する無駄な連言照合は行われない．したがって，generator 節で行われる連言照合，および生成されたアトムを格納するためのメモリ量は，それぞれ基本アルゴリズムの場合の $O(m^4)$ から $O(m^2)$ に減少する．

4.3 遅延アルゴリズム

遅延アルゴリズムでは，tester 節から要求があるたびに毎回 1 個ずつ新アトム Δ が generator 節によって生成され，テスト済みのアトム集合（全テストアルゴリズムにおける $M \cup D$ と等価）に対して，この新アトムの棄却テストと包摂テストが行われる．

遅延アルゴリズムは要求に応じて新アトムを生成するので，基本アルゴリズムのような過剰生成による無駄はない．また，新アトムについては直ちに棄却テストを行っているため，棄却アトム X を生成した時点で偽が判明する．したがって，全テストアルゴリズムと同様の効果が生じ，遅延アルゴリズムの複雑さは，オーダ的には図6の全テストアルゴリズムと同じになる．ただし，棄却アトム X を生みだすモデル拡張 $CJM_{GS}(\{a_m\}, M)$ は，全テストアルゴリズムでは一度に行われるのに対し，遅延アルゴリズムでは要求に応じて徐々に行われるので，X がどの連言照合操作によって生まれるかで，最大 $2m$ 回分の連言照合・包摂テストが省ける．

遅延アルゴリズムと全テストアルゴリズムの複雑さのオーダが同じであるのは，全テストアルゴリズムにおいては，モデル拡張結果 *new* に対するテストの完了を待って次のモデル拡張を開始する，という逐次処理を前提としているからである．しかし，並列環境下で

* M の個数は $\lceil (m/\rho)^{1/2} \rceil$，$D$ の個数は $\lceil \rho m^2 \rceil - \lceil (m/\rho)^{1/2} \rceil$．
** 左の正方形の逆 L 字形の空白部分．

全テストアルゴリズムを実行した場合，適切なプロセス制御がなければ，generator プロセスが暴走して不要なアトムを多量に生成する危険がある．その結果，棄却アトム X が生成されてテストされる前に，膨大な数の無駄な包摂テストが行われてしまうことになる．遅延アルゴリズムは，このような並列環境下においても，generator プロセスの暴走を自然に防ぐことができ，generator 節における計算量とメモリ量のオーダを $O(m^2)$ に保つことが可能である．

4.4 単位負節に対する最適化

先読みと部分評価による最適化は，いずれも各アルゴリズムにおいてテストされるアトム集合より一世代先のアトム集合を調べるという先行テスト効果を持っている．

generator 節の前件リテラル数が 2 という仮定のもとでは，モデル拡張候補の要素数はモデル候補の要素数の 2 乗オーダになる．

基本アルゴリズムでは，モデル拡張候補に対する棄却テストは行われておらず，モデル候補に対してのみ行われる．これに先行テストを施すとモデル拡張候補に対する棄却テストが可能になり，generator 節の連言照合回数ならびにメモリ量のオーダを平方根分下げることができる．この結果，全テストおよび遅延アルゴリズムと同じ効果が得られる．この最適化は，とりもなおさず OTTER でとられているアルゴリズムそのものである．

一方，全テストと遅延アルゴリズムでは，モデル拡張候補に対するテストは完全に行われている．これに先行テストを施すとモデル拡張候補からさらに次の世代のアトム集合に対する棄却テストが可能になり，やはりこれらのアルゴリズムにおける generator 節の連言照合回数とメモリ量のオーダも平方根分下げることができる．

基本アルゴリズムを示す図 5 において，D の領域のアトムは最適化を施すことによって作られなくなるので，generator 節の計算量およびメモリ量のオーダは $O(m^2)$ になる．

全テスト/遅延アルゴリズムを示す図 6 において，最適化後は，M の領域のアトムが生成された段階で D の領域のアトムに対する棄却テストも完了するので，generator 節の計算量およびメモリ量のオーダは $O(m)$ まで減少する(**図 7** の先読み最適化の効果参照)．

部分評価による最適化では(**図 8** 参照)，元の単位負節の代わりに部分評価によって得られる 2 リテラル負節を用いて棄却テストを行うが，これは先読み最適化

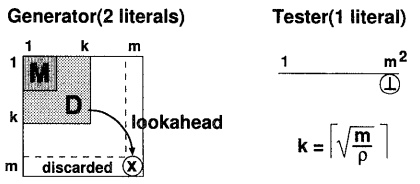

図 7 全テスト/遅延における先読み最適化の効果
Fig. 7 Effect of the lookahead optimization in the full-test/lazy algorithms.

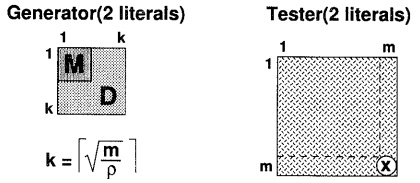

図 8 全テスト/遅延における部分評価の効果
Fig. 8 Effect of partial evaluation in the full-test/lazy algorithms.

において new_{next} を生成し，単位負節によってテストすることに相当する．また，図 7 および 8 において，M で示された領域はどちらの最適化手法をとっても再度計算される．したがって，先読み方式と部分評価方式は，計算量とメモリ量のオーダという点では等価な最適化手法である．

4.5 複雑さの解析結果の要約

以上述べてきたアルゴリズムの複雑さの議論をまとめると**表 1** のようになる．T, S, G はそれぞれ棄却テスト，包摂テスト，モデル拡張に要する計算量を，M は必要となるメモリ量を表す．表中 α は包摂テストの効率を表す($1 \leq \alpha \leq 2$)．$\alpha=1$ のときは，ハッシング効果が高く定数オーダで包摂テストが行えることを意味し，$\alpha=2$ のときは，リストに対する線形探索の場合のように，要素数に比例する手間で包摂テストを行うことを意味する．

基本，全テスト/遅延，全テスト/遅延＋最適化といくにしたがって，必要メモリ量は着実に平方根ずつ減少している．これは生成されるアトム数が減少することを意味しており，それに応じて包摂テスト回数も同様に減少する．$\alpha=2$ の場合を考えると，最も高価な計算は包摂テストであるから，その計算量の削減は全体の計算量の削減につながる．一方 $\alpha=1$ の場合には，2 リテラル負節の棄却テストが最も高価な計算となるが，これはいずれのアルゴリズムでも一定であり，遅延計算による改善効果は定数倍の速度向上に留まる．いずれにしても，遅延計算を採用することにより，全計算量は棄却テストの計算量で抑えられるようになる

表1 各アルゴリズムの計算量
Table 1 Summary of complexity analysis.

	単位負節			
	T	S	G	M
基本	ρm^2	$\mu\rho^2 m^{4\alpha}$	$\rho^2 m^4$	$\rho^3 m^4$
全テスト / 遅延	ρm^2	$\mu m^{2\alpha}$	m^2	ρm^2
+ 最適化	m^2	$(\mu/\rho)m^{\alpha}$	m/ρ	m

	2 リテラル負節			
	T	S	G	M
基本	$\rho^2 m^4$	$\mu\rho^2 m^{4\alpha}$	$\rho^2 m^4$	$\rho^3 m^4$
全テスト / 遅延	$\rho^2 m^4$	$\mu m^{2\alpha}$	m^2	ρm^2

† m は 基本アルゴリズムにおいて false が検出された時点での モデル候補の要素数
‡ ρ は生成するアトムの生存率，μ は連言照合成功率 ($\rho \leq \mu$)，α は包摂テストの効率

ことがわかる．

5. 実験結果

5.1 広い探索空間を持つホーン問題

これまで述べてきたモデル生成アルゴリズムの評価実験には，米国アルゴンヌ国立研究所の McCune らが，彼らの開発した汎用定理証明器 OTTER を使って挑戦中の condensed detachment 問題[4] を用いた．これらの問題は ⊃, ≡, ∨ や群の演算子等に関する1本の分離規則(detachment (modus ponens) rule)といくつかの公理および解きたい定理(負節で表現)からなり，すべて以下のようにホーン節のみの集合で記述される．

#28(i: implication/n: negation)

$p(X), p(i(X, Y)) \rightarrow p(Y)$.

$true \rightarrow p(i(i(i(X, Y), Z), i(Y, Z)))$.

$true \rightarrow p(i(i(i(X, Y), Z), i(n(X), Z)))$.

$true \rightarrow p(i(i(n(X), Z), i(i(Y, Z), i(i(X, Y), Z))))$.

$p(i(i(a, b), i(i(b, c), i(a, c)))) \rightarrow false$.

#64(i: implication)

$p(X), p(i(X, Y)) \rightarrow p(Y)$.

$true \rightarrow p(i(i(i(X, Y), Z), i(i(Z, X), i(U, X))))$.

$p(i(a, a)) \rightarrow false$.

上記の問題#28は，アトムの生存率 ρ が高く，モデル候補の要素数が急速に増加する傾向を示す例題であり，一方#64は，逆に ρ が低く，なかなかモデル候補が成長しない例題である．

Condensed detachment 問題は，記述量は少ないにもかかわらず計算量が膨大となり，定理証明器の良否を決定する際の重要な要素である推論速度や戦略を比較評価する上で格好のベンチマークである．

これらの問題を解くには，出現検査付きの完全な単一化が必要であり，さらに探索空間が一般に広いため，項の重みづけによる優先制御や削除のような様々なヒューリスティクスが必要となる．また幅優先と深さ優先の各戦略を適宜切替えるような制御機構も必要である．

5.2 アルゴリズムの性能比較

実験結果を**表2, 3**に示す．ここでは，削除戦略として項の大きさによる足切りおよびトートロジー除去のみを用いた．

各アルゴリズムはすべて KL1 で実装し，並列推論マシン PIM/m[8] の1 PE(Processor Element)上で測定した．各欄は，基本，全テスト，遅延の各アルゴリズムを示し，各行は，問題を解くのに要した時間，そのときに行った単一化と包摂テストの回数，必要としたメモリ量，連言照合の成功率(μ)とアトムの生存率(ρ)を示す．各行はさらに3行からなっており，上から単位負節に対する最適化を行っていない場合，先読み最適化を採り入れた場合(L)，部分評価による最適化を採り入れた場合(P)を表示している．単一化の回数では，generator 節(G)と tester 節(T)の連言照合で行われる単一化を分けて示している．なお，FAIL はメモリ不足(上限は $|M|+|D|$ が約10万)で解けなかったことを表す．

この実験結果は，表1で示されている計算量を如実に反映したものになっている．例えば，#28を部分評価なし(単位負節のまま)に解こうとすると，基本アルゴリズムではメモリ不足で解くことができないが，全テストおよび遅延アルゴリズムはメモリ量約1万2千で解くことができる．このように，アトム生存率 ρ の高い#28のような問題では，全テスト/遅延効果が顕著に現れる．これは，基本アルゴリズムの場合，テスト未完のまま蓄えられているアトム数($|D|$)が膨大になるからである．また，ρ の低い#64のような問題でも，必要メモリ量は確実に減っていることに注目されたい．

基本アルゴリズムに先読み最適化を採り入れたもの(基本+L)は，単位負節については全テストを行っていることになるので，condensed detachment 問題のような負節が単位節のみからなるような問題については，全テスト/遅延アルゴリズムと等価になる．また，OTTER では，単位節については特別な処理を行っており，単位節が生成された時点で直ちに単位反駁を試みる．これは，単位節については全テストを行ってい

表2 性能に関する結果(問題# 28)
Table 2 Results (Prob. # 28).

問題 # 28		基本	全テスト	遅延
時間 (秒)	-	FAIL	1912.42	1849.19
	L	1901.19	358.32	356.00
	P	2024.54	196.62	198.67
単一化 G+T (回)	-	FAIL	672399+12292	671920+12292
	L	672399+12292	24179+1059753	24064+1059753
	P	671579+362853	24179+364416	24064+364416
包摂テスト (回)	-	FAIL	338326	338226
	L	338326	13358	13300
	P	338106	13358	13300
メモリ量 $\|M\|+\|D\|$ (個)	-	FAIL	819+11476	819+11473
	L	819+11473	154+668	154+665
	P	819+11470	154+668	154+665
μ/ρ	-	FAIL	0.577/0.018	0.577/0.018
	L	0.577/0.018	0.633/0.035	0.633/0.035
	P	0.577/0.018	0.633/0.035	0.633/0.035

表3 性能に関する結果(問題# 64)
Table 3 Results (Prob. # 64).

問題 # 64		基本	全テスト	遅延
時間 (秒)	-	277.65	211.28	208.73
	L	209.54	192.32	190.94
	P	230.50	165.31	163.77
単一化 G+T (回)	-	93329+305	70755+306	70715+305
	L	70755+306	50624+137351	50465+137351
	P	70489+44960	50624+45423	50465+45423
包摂テスト (回)	-	81419	62602	62576
	L	62602	44336	44324
	P	62443	44336	44324
メモリ量 $\|M\|+\|D\|$ (個)	-	305+112	265+41	265+40
	L	265+41	224+50	224+41
	P	265+39	224+50	224+41
μ/ρ	-	0.929/0.004	0.946/0.004	0.945/0.004
	L	0.946/0.004	0.947/0.005	0.944/0.005
	P	0.943/0.004	0.947/0.005	0.944/0.005

ることになり，OTTER と基本+L もやはり等価になる．

しかしながら，OTTER では，非単位節については特別な処理を行っているわけではないので，部分評価による最適化は，基本アルゴリズムにこの最適化を施したのと同じく無効である．

一方，全テスト/遅延アルゴリズムでは，単位負節に限らず，すべての負節について全テストを行っているので，先読みや部分評価による最適化によってさらに効率が改善される．例えば# 28 では，メモリ量が約 12000 から 800，時間が 1900 秒から 200〜350 秒に改善されている．

オーダ的には先読み最適化と部分評価による最適化は同じであるが，実験結果では差異が現れている．# 28 では，遅延+部分評価(P)の場合 198.67 秒，遅延+先読み(L)の場合 356.00 秒であった．この差は，tester 節で行われる単一化回数の差である．この理由は，遅延+先読みでは，generator 節 $p(X), p(i(X, Y))\rightarrow p(Y)$ によってアトムが作られた後，単位負節 $p(A)\rightarrow$ *false* との照合が行われるのに対し，部分評価による最適化では，A の束縛情報が 2 リテラル負節 $p(X), p(i(X, A))\rightarrow$ *false* の前件部に伝播し，照合の失敗が早めに検出されるためである．

以上示したように，全テスト/遅延アルゴリズムによりメモリ量の削減が計られ，それに伴い，最も計算量を占める包摂テストの量が削減され，実行時間が格段に向上していることがわかる．

5.3 MGTP と OTTER の性能比

本節では condensed detachment 問題 112 題について，KL1 で実装された MGTP(PIM/m-1PE 上)と C で実装された OTTER(SPACE II 上)の性能を比較する．

Condensed detachment 問題を解く上での非決定性は，モデル拡張候補(D)からの要素 Δ の取り出し方にある．計算量の解析では，この取り出しは，要素の生成順，つまり D は FIFO のキューであることを仮定した．前節の評価で用いた MGTP でも，取り出しはキュー方式を採っている．一方，OTTER では D の中から最も重みの軽い要素を Δ として取り出すソーティング方式を標準としているが，このほかにも，キュー方式との併用など様々な取り出し方を指定することができる．

5.3.1 キュー方式での比較

MGTP, OTTER ともにキュー方式で証明を行った場合の性能を述べる．ただし，MGTP は遅延アルゴリズムを用い，単位負節に対する部分評価の最適化を行っている(前節で述べたように OTTER では，この最適化は無効)．この場合，OTTER では 112 題中 57 題を，MGTP ではこれらを含めた 88 題を解くことができた．MGTP で解けて OTTER で解けなかった 31 題のほとんどは，メモリオーバフロー(15 MB 超)が原因であった．両者が解けた 57 題についての証明速度比は，OTTER を 1 としたとき MGTP は 0.3〜22 である．

参考までに，実行時間の約 9 割を占める単一化および包摂テストの速度を比較すると，問題により差異はあるが，MGTP はそれぞれ，OTTER の平均 0.25 および 0.15 倍である．このように，基本手続きの遅さにもかかわらず，証明速度が最大 20 倍程度向上している理由は，遅延ならびに最適化による計算量削減効果によるものである．MGTP の実行速度の方が遅かった問題は，前節の# 64 のようにアトム生存率が低く，計算量の改善効果が少ない問題である．この場合，基本手

続きの遅さが証明速度に直接反映し，OTTERより約3倍程度遅くなっている．しかし，メモリオーバフローのためOTTERで解けなかった問題も解けるようになっていることから，遅延ならびに最適化のメモリ量削減効果も多大であることがわかる．

5.3.2 キュー方式 v.s. ソーティング方式

OTTERでは，本論文で述べたアルゴリズムの改良の代わりに，ソーティング方式を用いることによって，計算量とメモリ量の改善効果を狙っている．これは，より単純な(シンボル数が少ない)項ほど有用である，というヒューリスティクスに基づいているが，確率的なものであり，効果が現れるかどうかは問題による．

ソーティング方式のOTTERを用いると，112題中94題を解くことができるようになり，解ける問題数に関してはキュー方式のMGTPとほぼ同等になる．このうち，キュー方式で解けずソーティング方式で解けるようになった問題は18題，逆にソーティング方式で解けずキュー方式で解ける問題は13題であった．両方の方式で解ける76題の問題のうち，24題はキュー方式の方が証明時間が短く，35題はソーティング方式の方が短い(残り17題はほぼ同じ)．

このように，両方式ともそれぞれが優位であった問題数はほぼ同等であり，定理証明システムにとって，ソーティングも重要な機能であることがわかる．今後この機能をMGTPに採り入れていく予定であるが，ソーティングは生成し得るアトム全体を対象にするという点で，アトム生成量に歯止めをかける遅延という考え方と相反する．そこで，遅延と同等の効果を持つ全テストアルゴリズムにソーティングを組み込むことを検討している．

6. 結　論

ボトムアップ定理証明器上で深い推論を要する定理を証明する際には，計算量およびメモリ空間の爆発を防ぐことが肝要であり，その解決法として，我々は遅延モデル生成という考えに基づき，いくつかの効率改善アルゴリズムを示した．

逐次マシン上で実行する場合，計算量およびメモリ量のオーダに関しては，本論文で述べた全テストアルゴリズムは遅延アルゴリズムと同様の効果があるので，全テストアルゴリズムを採用すれば十分である．しかし並列環境下では，生成プロセスとテストプロセスの間の同期をうまくとらないかぎり，全テストアルゴリズムは無駄なアトム生成ならびに包摂テストを行ってしまう危険がある．一方，遅延アルゴリズムは要求駆動の概念に基づいており，かかる制御が自然に行えるためこのような危険がなく，並列マシン上で実行する場合に最も効果を発揮する．

ハードな数学的定理証明問題として知られているcondensed detachment問題を対象とした実験結果から，遅延アルゴリズムに加えて単位負節に対する最適化技法を用いることにより，計算量およびメモリ空間が著しく削減できることが実証された．また，SPARC II上のOTTERとの比較評価の結果，本アルゴリズムを採用したPIM/m-1PE上のMGTPは，OTTERとほぼ同等か問題によっては1桁以上も上回るという良好な結果が得られた．

遅延モデル生成の考えはホーン節集合から非ホーン節集合に容易に拡張でき，また，超導出やset-of-support戦略を使った他の一般のボトムアップ定理証明器にも適用可能である．本論文では'generate-only-at-testing'を実現したが，より先進的な概念である'generate-only-for-testing'の方がさらに重要である．これは*magic sets*や*relevancy testing*[9]のような探索空間の刈り込み戦略の採用を意味するが，これらの戦略の制御構造は非常に単純であり，現在の枠組にうまく組み込めると思われる．

我々は現在遅延モデル生成法に基づく並列定理証明器を開発中であり，この並列化方式ならびに並列性能については次の論文で報告する予定である．

参考文献

1) Ueda, K. and Chikayama, T.: Design of the Kernel Language for the Parallel Inference Machine, *Computer Journal*, Vol. 33, No. 6, pp. 494-500 (1990).
2) Manthey, R. and Bry, F.: SATCHMO: A Theorem Prover Implemented in Prolog, *Proc. of CADE-88*, pp. 415-434 (1988).
3) Łukasiewicz, J.: *Elements of Mathematical Logic*, Pergamon Press, Oxford (1963).
4) McCune, W. W. and Wos, L.: Experiments in Automated Deduction with Condensed Detachment, *Proc. of CADE-92*, pp. 209-223 (1992).
5) McCune, W. W.: *OTTER 2.0 Users Guide*, Argonne National Laboratory (1990).
6) Hasegawa, R., Koshimura, M. and Fujita, H.: Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers, *Proc. of the IFIP TC12/WG12.3 International Workshop on Automated Reasoning*, Beijing, pp. 221-238 (1992).
7) Fujita, H. and Hasegawa, R.: A Model-Generation Theorem Prover in KL1 Using Ramified Stack Algorithm, *Proc. of the Eighth*

International Conference on Logic Programming, pp. 535-548, The MIT Press (1991).
8) Nakashima, H. et al.: Architecture and Implementation of PIM/m, *Proc. of the International Conference on Fifth Generation Computer Systems*, pp. 425-435 (1992).
9) Wilson, D. S. and Loveland, D. W.: Incorporating Relevancy Testing in SATCHMO, CS-1989-24, Department of Computer Science, Duke University, Durham, North Carolina (1989).

(平成6年1月26日受付)
(平成6年12月5日採録)

長谷川隆三（正会員）

1949年生. 1972年九州大学工学部通信工学科卒業. 1974年九州大学大学院工学研究科通信工学専攻修士過程修了. 同年日本電信電話公社入社. 同社武蔵野電気通信研究所勤務. 1987年(財)新世代コンピュータ技術開発機構へ出向, 現在に至る. 九州大学博士(工学). ポリプロセッサシステム, データフローマシン, 関数型言語, 論理プログラミングおよび定理証明に関する研究に従事. 電子情報通信学会会員.

越村　三幸（正会員）

1961年生. 1986年筑波大学第一学群自然学類卒業. 1986年同大学院修士過程理工学研究科修了. 同年日本ビジネスオートメーション(株)（現東芝情報システム(株)）入社. 現在同社オープンシステム本部所属. 1986～1990年, 1993年(株)新世代コンピュータ技術開発機構に出向. 並列オペレーティングシステム, 並列定理証明システムの研究開発に従事. 日本ソフトウェア科学会, 人工知能学会各会員.

藤田　博

1955年生. 1978年東京大学理学部物理学科卒業. 1980年同大学院情報工学修士課程修了. 同年三菱電機(株)入社. 同社中央研究所勤務. 1986年-1990年(財)新世代コンピュータ技術開発機構に出向. 工学博士. 論理プログラミング, 部分計算および定理証明に関する研究に従事. 日本ソフトウェア科学会, ACM各会員.