最小不動点計算に基づくプログラムの帰納的性質の導出†

小川瑞史†† 小野 諭†††

本論文では，関数型プログラムの帰納的性質を最小不動点計算に基づき求める手法について論じる．この手法は，検証したい性質に対応して有限抽象領域を定め，自動生成系により広域解析プログラム（最小不動点計算系）を生成する．そして，解析対象プログラムの抽象領域上の最小不動点を求め，性質を検証する．本手法は，検証系の停止性が保証され全自動の機械的検証が可能なほか，検証系の自動生成が可能という特徴を持つ．

1. はじめに

再帰的に定義された関数型プログラムの性質を検証する場合，しばしば帰納法が重要な役割を果たす．たとえば，自然数 N 上の関数 $f(n)$ が性質 ϕ を満たすことを主張する場合，まず初期条件 $\phi(f(0))$ を証明する．次に，$\phi(f(n))$ が $n=n_0$ または $n\leqslant n_0$ で成立しているという帰納法の仮定を置き，$\phi(f(n))$ が $n=n_0+1$ でも成立することを証明する（帰納ステップ）．この結果，数学的帰納法により $\forall n\in N[\phi(f(n))]$ が恒真であることがいえる．

プログラムの停止性を検証する帰納法としては整礎集合帰納法[3]，等価性を検証する帰納法としては構造帰納法[3),4)]，や計算帰納法[3),4)]，また項書き換え系の完備化アルゴリズムに基づくインダクションレス・インダクション[10]などが知られている．しかし，これらの帰納法の適用する変数の選択は発見的であり，一般に検証アルゴリズムの停止性も保証されない．実際，構造帰納法に基づく Boyer-Moore の自動証明系[2]なども，その使用に際し，インプリメントされた証明戦略への理解や，適切な補題の設定に基づくガイドが不可欠であり，また，インダクションレス・インダクションもプログラムの停止性を保証する構文的順序が与えられた場合にのみ有効である．

これに対し，検証の対象を，より簡単な性質である，計算出力の属性の判定（偶数/奇数の判定など），計算入力の属性の判定（ストリクト性の検出など），エラーを引き起こす可能性のある変則性の検出[7),14)]，などを対象とし，検証アルゴリズムの停止性を保証するデータフロー解析に基づき解析/検証を行うアプローチがある．この手法は，検出したい性質に対応した適切な有限抽象領域を与え，その上で最小不動点計算に基づきプログラムの抽象的実行をすべての可能な入力に関して行う．これにより，プログラムの近似的性質を得る．これを抽象的解釈[1]とよぶ．しかし，従来の抽象的解釈のフレームワークでは，リストなど再帰的に定義されるデータ構造をもつ領域上のプログラムを考えたとき，データ構造に付随する帰納的性質に対しては十分な解析/検証力を持たなかった．たとえば，自然数のリスト

$$\mathrm{List}=\mathrm{nil}+\mathrm{Int}\times\mathrm{List}$$

におけるヘッドストリクト性[11]

$$\text{Head-strict}=\perp+\text{strict}\times\text{Head-strict}$$

すなわち，リスト構造の評価と各リスト内の元の評価が同期的である性質は，抽象的解釈による検出はできないと考えられていた[9),11)]．

本論文では，すでに著者らが提案している計算経路解析[15]に基づき，拡張したモード付き計算経路解析[12),16)]を提案し，帰納的推論を可能とする抽象領域の設計のスキーマについて論じる．その際，帰納的ステップにおける推論は，抽象領域内での lub (least upper bound) 演算により行う．これを抽象実行帰納法とよぶ．この応用としてテール/トータル/ヘッドストリクト性の検証法を示す[13]．

対象とするプログラムは，再帰的データ構造を含む領域上の一階の関数型プログラムである．本手法は，真の性質を見逃す可能性がある，という意味で近似的検証法であるが，検証アルゴリズムの停止性を保証でき全自動の機械的検証が可能，抽象領域の仕様に基づき検証プログラムの自動合成が可能などの利点をもつ．

本論文では，さらに，抽象実行帰納法の自動生成システム[17]について報告する．本システムは，検出対象となる性質に対応した抽象領域の仕様（CPA のモー

† Deriving Inductive Properties of Recursive Programs Based on Least-fixpoint Computation by MIZUHITO OGAWA (NTT Basic Research Laboratories) and SATOSHI ONO (NTT Software Laboratories).

†† NTT 基礎研究所
††† NTT ソフトウェア研究所

ド定義）を表の形で与え，それに基づき，一般不動点計算系と合成することにより，広域解析プログラムを自動生成する．これに解析対象プログラムを与え，結果として得られたモード領域上の関数としての最小不動点により，予測した性質が検証できたかを調べる．このような自動生成系を用いるのは，抽象実行帰納法では，検出する性質は近似的性質であるので，解析対象となるプログラムに応じ，必要なだけ解析の精度を上げるため，および，目的となる性質に応じ推論のスキーマを切り換えるためである．このような自動生成システムとして，既に Cecil[8] があるが，Cecil は手続きの順序の誤りなど変則性の検出を目的とし，逐次的プログラムを対象としたローカルな性質を検出する関数内解析を生成する．本システムは，計算の因果関係のみに基づく宣言的性質を求めるため並列処理にむく，およびグローバルな性質を検出する関数間解析を生成する，の二点において優れている．

2. 抽象的解釈と計算経路解析

2.1 抽象的解釈

抽象的解釈 (abstract interpretation)[1] は，ストリクトネス解析，副作用解析など関数間にわたる広域解析を，抽象（有限）領域上のプログラムの実行として定式化する手法である．すなわち，プログラムを解析するには，すべての入力に対し実行すればその性質がわかるが，これは計算不能である．抽象的解釈は，そのかわりに，対象とする性質を反映した有限領域上ですべての入力に対し，最小不動点計算に基づき実行することにより，近似的にプログラムの性質を解析するのが，広域解析であるととらえる．たとえば，計算結果が正になるか，負になるかを調べるためには，有限領域として $\{+,-\}$ (i.e. $x \geq 0$ なる x を $+$，$x<0$ なる x を $-$とする）をとると，加算 add は，

$$\mathrm{add}(+,+)=\{+\},\ \mathrm{add}(-,-)=\{-\},$$
$$\mathrm{add}(+,-)=\mathrm{add}(-,+)=\{+,-\}$$

となる．これにより，正の数同士の和は正，負の数同士の和は負であることがわかるが，正の数と負の数の和は正か負か，わからずいずれの可能性もあり，解析の結果が近似的であることを示している．

2.2 計算経路解析

関数型プログラムの宣言的意味はデマンド駆動型の実行により実現される．計算経路解析 (Computation Path Anlysis, 以下，CPA と略す) [15] は，基本関数をもとにユーザが定義した再帰関数のデマンドの伝搬のパターンを検出する．たとえば，基本関数 if(x, y, z) に対するデマンドは変数 x, y, z に対し，$\{x, y\}$ または $\{x, z\}$ のパターンで伝搬する．これをもとに，たとえば，

```
(defun foo (x, y)
 (if (zerop x) 1 (1+ (foo (1- x) (foo y x)))))
```

としたとき，foo (x, y) へのデマンドは変数 x にのみ伝搬することが CPA により解析される[15]．（以下，同様にプログラムのシンタックスは lisp に準じる．）

CPA は抽象的解釈により，有限領域 Abs$=\{\alpha^1, \alpha^\perp\}$ 上のプログラムの実行として定式化される[11]．ここで，α^1 は evaluated, $\alpha^\perp$ は unevaluated を意味する．より詳細には，関数 f に対し逆写像 f^{-1} を考え，f の出力を得るのに必要な最小限の入力の組を構成する．これを抽象領域 Abs 上の関数に射影する．この意味で，CPA は逆方向型解析 (backward analysis)[1] である．この過程は，たとえば，sor$_3(x, y, z)$（3変数の serial-or）では，未定義値を$\perp$で表すと，

$$\mathrm{Min}\ (\mathrm{sor}_3^{-1}(\mathrm{true})) \rightarrow \{(\mathrm{true}, \perp, \perp), (\mathrm{false}, \mathrm{true}, \perp), (\mathrm{false}, \mathrm{false}, \mathrm{true})\}$$
$$\mathrm{Min}\ (\mathrm{sor}_3^{-1}(\mathrm{false})) \rightarrow \{(\mathrm{false}, \mathrm{false}, \mathrm{false})\}$$
$$\mathrm{Min}\ (\mathrm{sor}_3^{-1}(\perp)) \rightarrow \{(\perp, \perp, \perp)\}$$

から，proj: true, false$\rightarrow\alpha^1$, $\perp\rightarrow\alpha^\perp$ により，

$$\mathrm{sor}_3^{\mathrm{CPA}}(\alpha^1) \rightarrow \{(\alpha^1, \alpha^\perp, \alpha^\perp), (\alpha^1, \alpha^1, \alpha^\perp), (\alpha^1, \alpha^1, \alpha^1)\}$$
$$\mathrm{sor}_3^{\mathrm{CPA}}(\alpha^\perp) \rightarrow \{(\alpha^\perp, \alpha^\perp, \alpha^\perp)\}$$

が導かれる．任意の関数 f に対し，$f(x_1, \cdots, x_n)$ にデマンドが伝搬しなければ $x_1, \cdots, x_n$ にも伝搬しないので，$f^{\mathrm{CPA}}(\alpha^\perp) \rightarrow \{(\alpha^\perp, \cdots, \alpha^\perp)\}$ となる．したがって，デマンドの伝搬パターンは $f^{\mathrm{CPA}}(\alpha^1)$ のみで表される．実際，sor$_3^{\mathrm{CPA}}(\alpha^1)$ より，sor$_3$ のデマンドの伝搬パターンは $\{\{x\}, \{x, y\}, \{x, y, z\}\}$ となる．

また，このパターンすべての共通部分がストリクトな変数 (e.g. x)，すべての結合がレレバントな変数 (すなわち，不要ではない変数，e.g. x, y, z) である．

2.3 モード付き計算経路解析

前節においては，自然数やブール値などデータ構造をもたないフラットな領域上のプログラムを対象としていた．しかし，再帰的なデータ構造をもつリストなどノンフラットな領域におけるデータ構造に付随した性質を解析するには，抽象領域が Abs$=\{\alpha^1, \alpha^\perp\}$ では十分でない．たとえば，本論文の以下の章であつかうヘッド/テール/トータルストリクト性[9),11),13)] などの

検出がその例となる.

リストにおいては,たとえば,テールストリクトネス解析の場合, Abs=$\{\alpha^{\perp}, \alpha^{1}, \alpha^{2}, \alpha^{nil}, \alpha^{*}\}$ などのように抽象領域を設定する. ここで, $\alpha^{\perp}$ は unevaluated, α^{1} は possibly infinite lists (リストの末尾が unevaluated), α^{2} は finite lists(リストの末尾が evaluated), α^{nil} は nil, α^{*} は inconsistent を意味する. ここで, finite lists を α^{2}(non-nil) と α^{nil}(nil) にわけたのは, リストの計算においては, 一般に nil が再帰の初期条件となるため, それに対応している. この初期条件の条件分岐の判定にともない, ブール値はもともと有限領域であるので抽象化せず, α^{true}, α^{false} として設定する.

このため, フラットな領域の場合は $(\alpha^{1}, \alpha^{\perp}) \Leftrightarrow \{x\}$ や $(\alpha^{1}, \alpha^{1}) \Leftrightarrow \{x, y\}$ などのように, デマンドが伝搬する変数名のみに注目すれば伝搬パターンが示せたのに対し, $(\alpha^{nil}, \alpha^{\perp})$, (α^{2}, α^{1}) などのように, 伝搬モードを変数の組に対応するモードの組としてあらわすことが必要となる. このような拡張された CPA を, モード付き CPA (以下, 単に CPA といえば, モード付き CPA をさすものとする) とよぶ. たとえば, cons(x, y) が finite lists になるためには, x は評価する必要がなく, y が nil または finite lists でなければならないので,

$$\mathrm{cons}^{CPA}(\alpha^{2}) \rightarrow \{(\alpha^{\perp}, \alpha^{nil}), (\alpha^{\perp}, \alpha^{2})\}$$

となる. 理論的詳細については, 文献 12) を参照されたい.

3. 抽象実行帰納法における推論機構

3.1 抽象領域の設計と帰納法との関係

抽象実行帰納法は, 帰納的性質を反映した抽象領域上でのモード付き CPA により実現される. 直観的には, 毎関数展開ごとに分岐する各計算経路上における性質の遷移をすべての場合について, 最小不動点計算により追跡する. この性質を表現するものがモード (i. e. 抽象領域の元) である.

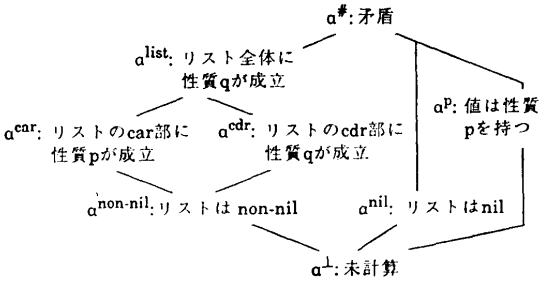
1章で説明したように, 帰納法は, 初期条件の成立の検証と, 帰納的仮定に基づく帰納ステップの2段階で行われる. このことは, モード付き CPA の抽象領域の設計に, 次のように反映される.

(1) 抽象領域に, 実領域の基底元 (帰納法の初期条件に対応する要素, たとえばリストでは nil) およびブール値と1対1に対応を持つ非抽象化元 (たとえば, α^{nil}, α^{true}, α^{false}) が存在すること.

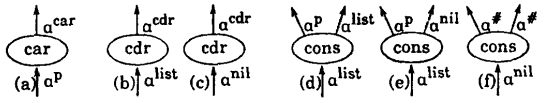
(2) 抽象領域が未計算を表す最小元 $\alpha^{\perp}$(: delay) と矛盾を表す最大元 α^{*}(: inconsistent) を持つ完備束になっていること.

(1)は初期条件成立の検出に, また, (2)は構造帰納法の帰納ステップに対応している. すなわち, 各ステップにおける推論は, 完備束における lub (least upper bound) をとる操作として表される.

抽象領域の lub 演算は, 同一変数が複数回参照されていて, 推論の結果, 異なった抽象元が割り当てられた場合に, 冗長な性質を削除する, 矛盾を検出する, より強い性質を導く, などのことを行う. これは, 異なる計算経路が合流したとき, それぞれが伝搬する性質から, それらの合成を推論することに相当する. たとえば, 実領域がリスト構造の場合, 抽象領域の構造や構成子, 分解子の推論のスキーマは, 典型的には**図1**のようになる. リストは, car 部と cdr 部の2要素を持つ. この例では, リストが性質 q を持つこと (e. g. α^{list}) は, car 部が性質 p を持つこと (e. g. α^{car}), および cdr 部が性質 q を持つこと (e. g. α^{cdr}) から推論される. したがって, 図1(1)のように, lub$(\alpha^{car}, \alpha^{cdr}) = \alpha^{list}$ なる性質を持つ領域が使用され, 長さが1大きいリストへ性質 q が波及される. 冗長な元の削除は, たとえば, α^{cdr} と α^{list} をそれぞれ伝搬する計算経路が合流したとき, lub$(\alpha^{cdr}, \alpha^{list}) = \alpha^{list}$ より α^{list} に整理されることに現れる. また, α^{nil} と α^{list} のように相反する性質が合流したときは, その lub 演算の結果, lub$(\alpha^{nil}, \alpha^{list}) = \alpha^{*}$ となり矛盾が導かれ, その計算経路は削除される.



(1) 抽象領域の構造のスキーマ
(1) Structure scheme for abstract domains.



(2) 関数領域の推論スキーマ
(2) Inference scheme for primitive functions.

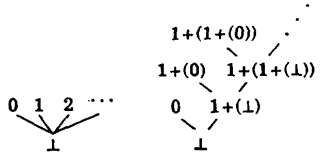
図1 リストに関する解析のデータ構造スキーマ
Fig. 1 Data structure scheme for list domains.

各基本関数は，図1(2)のような推論規則に対応する．ここで，(c) cdr および (e) cons 第二引数における α^{nil} は，それぞれ，初期条件として基底元 nil が性質 q を持つこと，および，(計算が停止する場合には) いつかはリストの性質 q は nil の場合に還元されることを示している．また，(f) cons における α^* では，cons の出力として nil が仮定されると矛盾が導かれることを示している．

3.2 抽象実行帰納法の適用例：Even/odd 解析

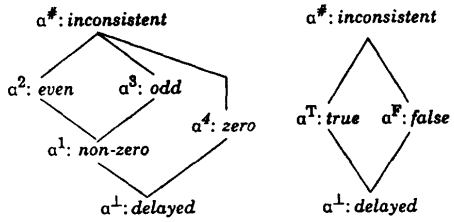
構造データは，一般に，構成子（自然数では 1+，リストでは cons)，分解子（1− や car, cdr)，これ以上分解できない基底元（0 や nil)，および基底元判定関数（zerop や null）からなる．また，条件関数 if では，第一引数のブール値で，動作が大きく異なる．これらに加え，定数である true, false, 0, nil を0引数関数とみなしたものを合わせて基本関数とする．定数を関数化するのは，CPA では，すべての計算が関数領域での合成で行われるためである．

本章では，以下でリストより簡単な構造をもつ再帰的に構成された自然数領域上で，奇数・偶数の判定をする Even/odd 解析を例に，抽象実行帰納法のアルゴリズムの概略を説明する．図2(b) のように再帰的に構成された自然数の領域は，図2 (a) に示す2.2節で取り上げた通常の自然数の領域と異なり，構成子 1+ を持つ．そのため，1+(⊥) (i.e. 1以上の自然数にあたる）のような遅延された自然数の表現を持つ．



(a) (通常の)自然数　(b) 遅延された自然数
(a) Integers.　(b) Lazy integers.

図 2 二つの自然数領域
Fig. 2 Two kinds of integer domains.



(a) 自然数型　(b) ブール型
(a) Integer type.　(b) Boolean type.

図 3 Even/odd 解析の抽象領域
Fig. 3 An abstract domain for Even/odd analysis.

Even/odd 解析に使用する抽象領域のハッセ図式は図3に示すとおりである．まず，抽象領域の元は自然数型とブール型に大別される．自然数型は，さらに0(: zero) と非0(: non-zero) に，非0は，正の偶数 (: even)，正の奇数 (: odd) へと分類される．ブール型は，真 (: true) と偽 (: false) からなる．この例では，元の上限 (lub) は，条件が同時に成立する場合の推測規則となっている．たとえば，ある元が : non-zero かつ : even である場合，その元の性質は : even であると推論される．また，ある元が : true かつ : false である時や，: odd かつ : zero である時など : inconsistent と推論される．

図4は，抽象領域での基本関数の計算例を示したものである．CPA のフレームワークでは，ある出力を得るためのすべての計算経路を逆方向にたどり，その結果，可能な入力の組をすべて求める．したがって，関数はすべて逆方向に実行されるので，入出力関係は，すべて実領域上の関数と逆になっている．図4の (a)，(b) は，条件関数で，第一引数がそれぞれ真，偽の場合に対応している．図中，〈属性〉は，任意の抽象元をさしていて，それがそのまま出力に波及することを表現している．(c, d) は基準元判定関数 zerop の例で，ブール値からその条件を満たす構造データを生成する．(e, f, g) は実領域での構成子 1+ の場合で，また (h, i) は実領域での分解子 1− の場合である．

3.3 抽象実行帰納法の実行例：Even/odd 解析

ここでは，図5で定義される関数のうち，再帰的に(再)定義された加算関数 lazy-add (x, y) などをプログラム例として，説明する．

初期条件成立の検出　図6は，lazy-add (: zero, : odd) が : odd になるという仮定に対応する抽象領域での計算経路を示したものである．図6 (a) において，条件関数は，: true が選択され，結果の属性

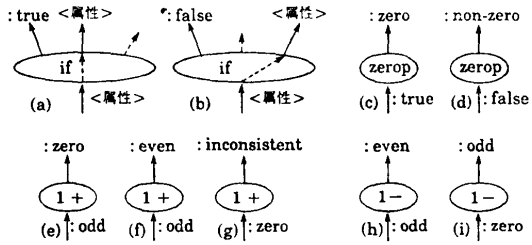
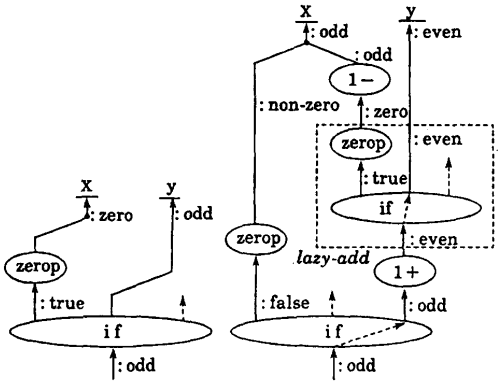


図 4 抽象領域における基本関数の逆方向実行例
Fig. 4 Actions of primitive functions on an abstract domain.

```
(defun lazy-add (x y)
  (if (zerop x) y
      (1+ (lazy-add (1- x) y))))
(defun lazy-mult (x y)
  (if (zerop x) (zero)
      (lazy-add y (lazy-mult (1- x) y))))
(defun evenp-1 (x)
  (if (zerop x) (true)
      (let ((y (1- x)))
        (if (zerop y) (false)
            (evenp-1 (1- y))))))
(defun evenp-of-n*inc-n (n)
  (evenp-1 (lazy-mult n (1+ n))))
```

図 5 Even/odd 解析の実例関数の定義
Fig. 5 Examples for Even/odd detection.



(a) 初期条件 (b) 帰納的ステップ
(a) Initial condition, (b) Inductive step.

図 6 Lazy-add(x, y)=: odd の検出
Fig. 6 Analysis on lazy-add(x, y)=: odd.

:odd から lazy-add の引数 x の属性 :zero および引数 y の属性 :odd が推論されていく様子は，明らかであろう．もし，条件関数で :false の経路を選択すると，図6 (b) のように，関数の再帰呼出し（点線内）を行う経路に入って行くことになる．

帰納ステップの実行 帰納ステップでは，非抽象化元に関する初期条件をもとに，構成子を適用したデータの性質を推論する．lazy-add の場合では，図6 (b) のように再帰呼出しに入ったとき，点線内の条件関数の第一引数が :true であるか :false であるかにより，同様に計算経路をたどる操作が行われる．図6 (b) は，:true を選択した場合であり，x は二つの経路より :non-zero と :odd の属性が，y は :even なる属性が伝搬される．さらに，x は lub 操作により，lub(:non-zero, :odd)⇒:odd のように冗長な属性である :non-zero が削除され :odd のみが導かれる．:false が選択された場合は，またさらに再帰呼出しの経路を初期条件に到達するまでたどる．このような操作を繰り返し，新しい入力の組が生成されなくなるまで繰り返す．形式的に，以上の操作は，有限完備束上の不動点計算として抽象的解釈により定式化することができる[12),13)]．したがって，アルゴリズムがある順序に関し単調であれば，アルゴリズムの停止性は保証される（参考文献 12) 参照)．

Even/odd 解析では，lazy-add の出力が奇数(:odd) になるのは，入力が偶数と奇数，または奇数と0などの場合であることを推論する．この結果を，さらに lazy-mult などへと波及させることにより，evenp-of-n*inc-n(n) が奇数を出力する計算経路がないことがわかり，『任意の自然数 n に対し，n と $n+1$ を掛けたものは偶数である』ことが示される．このことは，次章で自動生成システムの実行例として示す．

4. 抽象実行帰納法の自動生成系

抽象実行帰納法のシステムは，現在 VAX/VMS 上にその自動生成系が試作されている[17)]．プログラムは Common Lisp で記述され，約3K行程度，抽象領域を定義するテーブルが約 0.1K 行程度である．さらに，この試作システムは簡単な対話的環境を持っている．実行系は，**図7**に示すように，まず，求めたい性質から CPA のモード定義をユーザが表の形で与える．CPA のモード定義は抽象領域およびその上の各基本関数の抽象的実行の設計からなる．Even/odd 解析で使用する抽象領域を表すハッセ図式(図3)の lub 関係，および，各基本関数の対応する推論規則(図4)を表の形で与える．次に，自動生成系により，広域解析プログラム（最小不動点計算系）をその表に基づき生成し，これに解析対象プログラムを与える．結果として得られたモード領域上の関数としての最小不動点により，予測した性質が検証できたかを調べる．

evenp-of-n*inc-n(n) に対する Even/odd 解析の結果を**図8**に表す．行1～3では，解析定義データ，解析対象プログラムをロードし，解析系を生成している．行3の :fully-lazy-lub とは，解析生成系に lub 推論を行う解析系を生成すること，制御テーブル作成はすべて要求駆動で行うことを指示している．(要求駆動型生成の場合，解析系の生成は瞬時に終了するか，解析時間は若干長くなる.) 行5で，生成した解析系により，実際の不動点計算を行っている．表示の CPA 時間は，VAX 3500 上の VAX Common Lisp コンパイルドコードによる．

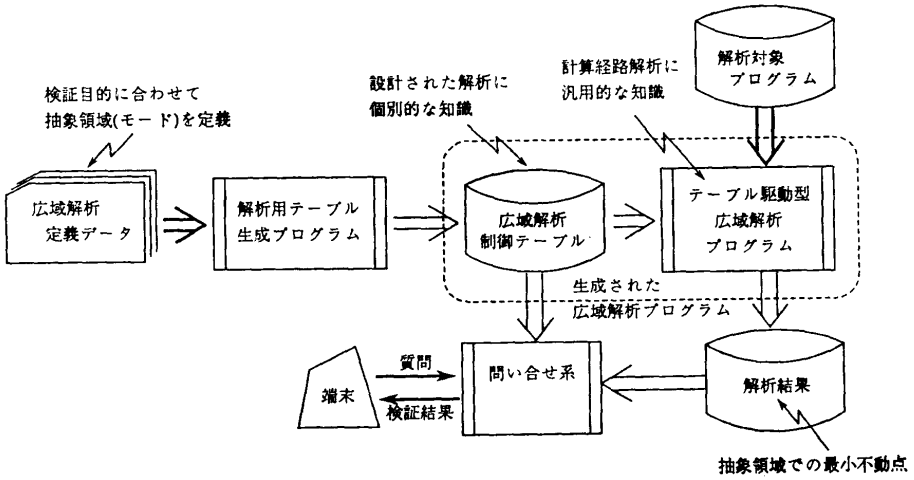


図 7 広域解析生成系に基づく検証システムの構成
Fig. 7 Verification system based on an automatic analyzer generator.

行8で，関数 evenp-of-*n**inc-*n* が，抽象領域の値 :true(すなわち，実領域のブール値 真)になる引数の組み合わせを問い合わせている．結果は，やはり抽象領域の値の組み合わせで表示される．答は，関数名，引数の数，可能な計算経路の並びで与えられる．((1 : zero)) とは，第一引数が抽象領域 :zero に属する場合に，値が :true になることを意味する．同様にして，この関数は，任意の自然数で値が真になることがわかる．一方，行10では，関数値が抽象領域の値 :false (実領域のブール値 偽) になる場合を調べているが，対応する計算経路は存在しない．これは，このようなことが起こりえないことを示している．これらの結果から，*n**(*n*+1) は，すべての自然数で偶数になることが導ける．

```
1. (load "primfunc-even-odd")          ;; loading CPA mode definition data
2. (load "funcs-for-even-odd")         ;; loading functions to be analyzed
3. (test-example-num :lub :fully-lazy-lub)     ;; generate analyzer
4.
5. (analyze '(lazy-add lazy-mult evenp-1 evenp-of-n*inc-n))
6. ;Converged after 6 iterations. CPU Time:37.94 sec., Real Time:38.07 sec.
7.
8. (show-all-possible-combinations 'evenp-of-n*inc-n :true)
9. (EVENP-OF-N*INC-N 1 ((1 :ZERO)) ((1 :ODD)) ((1 :EVEN)))
10. (show-all-possible-combinations 'evenp-of-n*inc-n :false)
11. (EVENP-OF-N*INC-N 1)
12. (show-all-possible-combinations 'lazy-add :odd)
13. (LAZY-ADD 2 ((1 :ZERO) (2 :ODD)) ((1 :EVEN) (2 :ODD))
                ((1 :ODD) (2 :EVEN)) ((1 :ODD) (2 :ZERO)))
```

図 8 Even/odd 解析における会話的応答
Fig. 8 Interactive session logging of the Even/odd analysis.

5. 抽象実行帰納法のスキーマ

5.1 帰納的性質の分類およびその例

Even-odd 解析は，lub 演算と計算経路の展開を行う順序にかかわらず解析結果はかわらない．この意味で，連続な解析とよばれる．しかし，リスト構造など，より複雑なデータ構造上の関数の帰納的性質を導くには，lub 演算と計算経路の展開を行う順序によっては，より弱い解析結果しか導けない非連続な場合がある．本章では，そのような非連続なスキーマが必要とされる場合について，リスト構造上のストリクト性解析を例に説明を行う．それらの検出する性質は，テールストリクト性[11),13)]，トータルストリクト性[9),13)]，ヘッドストリクト性[11),13)] である．自然数上のリストは，再帰方程式により，

$$\mathrm{list(Int)} = \mathrm{nil} + \mathrm{Int} \times \mathrm{list(Int)}$$

と表される．これに対応して，上にあげたストリクト性はそれぞれ，

テールストリクト性　tail = nil + delay × tail
トータルストリクト性　total = nil + strict × total
ヘッドストリクト性　head = delay + strict × head

として定義される．ただし，ここで delay はその引数に対し評価要求が伝搬されないことを意味し，strict はその引数に対し評価要求が伝搬されることを意味する．

直観的には，テールストリクトとは，結果を求める時に，リスト引数のトップレベル構造（スパイン）が定まる必要があることを，また，トータルストリクトとは，さらに各要素も定まる必要があることをいう．たとえば，リストの長さを求める関数 len-1(x) は，テールストリクトであり，リストの要素の合計を求める関数 sum-0 (x) は，トータルストリクトである．また，ヘッドストリクトとは，リスト引数のトップレベルで，構造と要素が同時に計算されていくことを示す．たとえば，リストの最初の要素から順に0を探し見つかれば真，見つからなければ偽を返す search-0 は，ヘッドストリクトである．

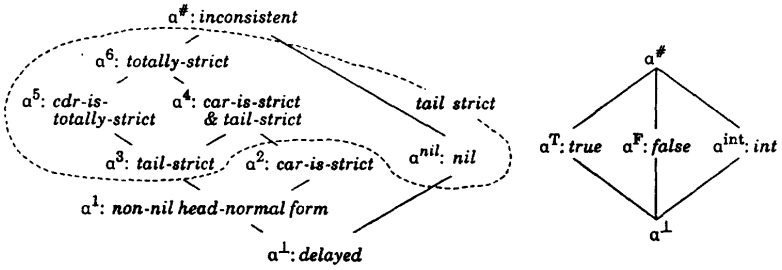


図 9 テイル/トータルストリクト性解析の抽象領域
Fig. 9 An abstract domain for tail/total strictness detection.

これらの解析のうち，テイルストリクト性検出のみ連続である．さらに，非連続な解析も，非連続・単調と非単調に分類される．次節以降で説明するように，トータルストリクト性検出は非連続・単調であり，ヘッドストリクト性解析は非単調である．

5.2 データ構造スキーマを用いた解析例（テイル/トータルストリクト性解析）

単調な性質とは，形式的には，ある性質 p が，抽象領域の元 x で成り立つ場合，任意の元 y ($y \sqsupseteq x \wedge y \not\cong \alpha^{\#}$) でも性質 p が成り立つことをいう．この場合，広域解析では，真の性質を下から近似すれば，完全な近似となる．これは，通常の解析と同様に，抽象領域における関数 f を単調にした (すなわち $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$) 最小不動点計算で実現でき，抽象領域が有限なので，解析の停止性も保証される．

テイル/トータルストリクト性は，単調な性質の例である．この解析の抽象領域を**図9**に示す．α^4, α^5, α^6 がそれぞれ，図1(1)のスキーマの α^{car}, α^{cdr}, α^{list} に対応していることに注意されたい．さて，テールストリクト性は連続であるが，トータルストリクト性は連続ではない[12]．（一般に，lub 演算と関数適用の順序が可換な時，連続という．i.e. $f(x \sqcup y) = f(x) \sqcup f(y)$.) したがって，$\alpha^{car}$, α^{cdr} から α^{list} を導く帰納ステップである lub 演算を再帰関数呼出しの計算経路を選択するたびに行わないと，トータルストリクト性は検出できない．

図 10 に，lub 演算を用いた場合と用いなかった場合の解析結果を比較した．ここで，**図 11** に示すとおり，関数 reverse-1, search-0, len-1, sum-1 は，それぞれ，リストのトップレベルを逆順にする関数，

```
    (1) With lub operation
(show-all-possible-combinations 'reverse-1 :totally-strict)
(REVERSE-1 1 ((1 :NIL)) ((1 :TOTALLY-STRICT)))
(show-all-possible-combinations 'search-0 :false)
(SEARCH-0  1 ((1 :NIL)) ((1 :TOTALLY-STRICT)))
(show-all-possible-combinations '(len-1 sum-1) :int)
((LEN-1 1 ((1 :NIL)) ((1 :TAIL-STRICT)))
 (SUM-1 1 ((1 :NIL)) ((1 :TOTALLY-STRICT))))
    (2) Without lub operation
(show-all-possible-combinations 'reverse-1 :totally-strict)
(REVERSE-1 1 ((1 :NIL)) ((1 :CAR-IS-STRICT :TAIL-STRICT)))
(show-all-possible-combinations 'search-0 :false)
(SEARCH-0  1 ((1 :NIL)) ((1 :CAR-IS-STRICT :TAIL-STRICT)))
(show-all-possible-combinations '(len-1 sum-1) :int)
((LEN-1 1 ((1 :NIL)) ((1 :TAIL-STRICT)))
 (SUM-1 1 ((1 :CAR-IS-STRICT :TAIL-STRICT)) ((1 :NIL))))
```

図 10 トータルストリクト性解析例 (lub 演算を用いる場合/用いない場合の比較)
Fig. 10 Total strictness analysis with/without lub.

```
(defun append-1 (x y)
  (if (null x) y
      (cons (car x)
            (append-1 (cdr x) y))))
(defun reverse-1 (x)
  (if (null x) (nil)
      (append-1 (reverse-1 (cdr x))
                (cons (car x) (nil)))))
(defun search-0 (x)
  (if (null x) (false)
      (if (zerop (car x)) (true)
          (search-0 (cdr x)))))
(defun len-1 (x)
  (if (null x) (zero)
      (1+ (len-1 (cdr x)))))
(defun sum-1 (x)
  (if (null x) (zero)
      (+ (car x) (sum-1 (cdr x)))))
```

図 11 ストリクト性解析の実例関数
Fig. 11 Definitions of example functions.

0を探索し見つかれば真を，なければ偽を返す関数，リストの長さを返す関数，リストのトップレベル要素を合計する関数である．

図 10 から，連続な性質であるテールストリクト性は lub 演算なしでも正しく検出されるが，非連続な性質であるトータルストリクト性は，lub 演算なしでは正しく検出できないことがわかる．これは，再帰的な関数定義においては，nil でないリストは car 部と cdr 部に分解される．このとき，毎関数展開ごとに lub 演算を用いてリスト全体の情報を正しく保持しないと，図1に表されている推論スキーマの限界を越えてしまい，トータルストリクト性の情報は復元できなくなってしまうためである．同時に，関数定義本体内で2回以上リストを分解している場合や，関数定義を unfolding[5] した場合には，トータルストリクト性は検出できない．

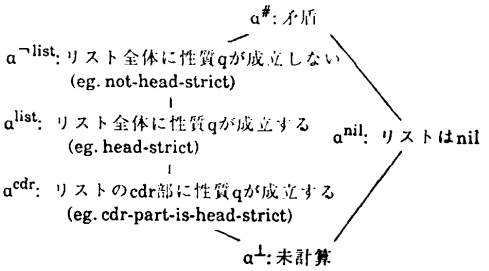
5.3 否定条件スキーマを用いた解析例 (ヘッドストリクト性解析)

ヘッドストリクト性検出は，否定条件を含むため，非単調な解析である．このような否定条件を含む解析のスキーマを**図 12** に示す．このとき，ヘッドストリクト性検出の抽象領域としては，α^{list} が head-strict, α^{cdr} が cdr-part-is-head-strict, $\alpha^{\neg list}$ が not-head-strict とそれぞれ対応する．それぞれのモードは，特別な場合として，α^{cdr} は cdr 部が未定義なリストを，α^{list} は car 部が性質 p (e.g. strict) を持ち cdr 部が未定義なリストを含んでいる．また，基本関数 cdr に対応する推論規則は $\alpha^{list}\to\alpha^{cdr}$, $\alpha^{cdr}\to\alpha^{\neg list}$ となるが，抽象領域上で lub 演算に従う順序$\sqsubseteq$に関し $\alpha^{cdr}\sqsubseteq\alpha^{list}\sqsubseteq\alpha^{\neg list}$ であるので，非単調性が現れている．

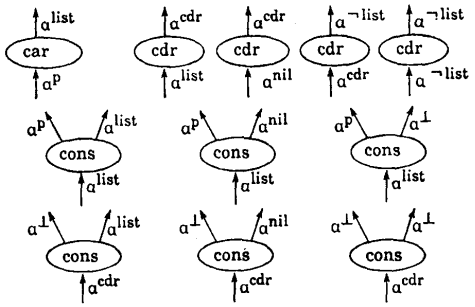
図 13 に append-0, search-0, reverse-0, sum-1 についての解析の結果を示した．これによると，前節の結果とあわせ，search-0 (x) は結果が false となるときは，トータルストリクトであり，結果が true になるときは，ヘッドストリクトであることがわかる．

非単調な解析は，従来法では解析の停止性を証明できない．しかし，α^0 と $\alpha^\#$ 以外の元全体を β とした時，抽象領域の基本関数が $\alpha^0\sqsubseteq\beta\sqsubseteq\alpha^\#$ の順序に関し単調な場合，不動点計算の繰り返しが集合包含関係に類した順序で単調なことがいえ，停止性を証明できる[12]．ヘッドストリクト解析もこの条件を満たしており，停止する解析である．

注意を要するのは，同一変数が複数の計算経路から head-strict および tail-strict の評価要求がなされていても，total-strict のみならず head-strict にすらなるとは限らないことである．(すなわち，lub (head-strict, tail-strict)⇒total-strict, head-strict.) 例えば，find-0 (x) をリストの先頭の要素から0を順に探し何番目に見つかったか計算する関数としたとき，find-0 (x)+len-1 (x) は第一の x は head-strict，第二の x は tail-strict と評価要求が伝搬するが，リスト x の最初の要素が0であるような場合，total-strict でも head-strict でもない．このように，否定条件スキーマは，通常有用な他の性質（例えば，テールストリクト性）ですら加わると容易に破壊される性質を扱っている．



(1) 抽象領域の構造のスキーマ
(1) Structure scheme for abstract domains.



(2) 関数領域の推論スキーマ
(2) Inference scheme for primitive functions.

図 12 リストに関する解析の否定条件スキーマ
Fig. 12 Negation scheme for a list domain.

```
(show-all-possible-combinations 'search-0 :true :minimal nil)
(SEARCH-0 1 ((1 :HEAD-STRICT)))
(show-all-possible-combinations 'search-0 :false :minimal nil)
(SEARCH-0 1 ((1 :NIL)) ((1 :HEAD-STRICT)))
(show-all-possible-combinations '(len-1 sum-1) :int :minimal nil)
((LEN-1 1 ((1 :NIL)) ((1 :CDR-IS-HEAD-STRICT)) ((1 :ANY-LISTS)))
 (SUM-1 1 ((1 :HEAD-STRICT)) ((1 :NIL))))
```

図 13 ヘッドストリクト性解析 (lub 演算を用いた場合)
Fig. 13 Head strictness analysis (with lub).

6. 抽象実行帰納法の限界および今後の課題

6.1 抽象実行帰納法の限界

抽象実行帰納法においては，(1)対象となる性質は単項的関係に限られる，および，(2)検出するのは近似的性質に限られ推論スキーマを越えた構造データの分解を行う再帰関数では検出力が著しく制限される，という限界がある．以下，それぞれについて論じる．

(1) 抽象実行帰納法における場合分けは，各変数それぞれに関し，抽象領域内の各元に対応して行われる．すなわち，帰納法の初期条件に対応しているかどうか，各データ構造内の構成子・分解子による各帰納ステップの要求条件を満たしているかどうか，などである．これらの条件は，あらかじめ与えられた抽象領域の定義により固定されているので，各変数ごとに定まる単項的関係（例えば，$x>0$ など）は検出できるが，複数の変数間で相対的に定まる多項的な関係（例えば，$x>y$ など）はほとんど検出できない．

(2) 推論スキーマとして，リスト上の関数に対し図7に示されるデータ構造スキーマを考える．たとえば，search-0 (x) は各関数展開ごとに，car, cdr により一度だけ入力リスト x を分解し，それぞれに処理を加え，また，cons により合成する．search-0′(x) を search-0 (x) を一度 unfolding[5] により展開した再帰関数定義をもつ関数とすると，実領域上での意味は変わらないが，トータルストリクト性などは検出できなくなる．これは，各関数展開時に car, cdr による分解が二度となり，推論スキーマの推論可能な範囲を越えてしまうためである．

6.2 他の解析との比較

抽象実行帰納法では，逆方向型の広域解析である計算経路解析 (CPA) をその基礎においている．同様に，逆方向型の解析で，テール/ヘッドストリクト性を検出する解析法として，projection transformer[11] がある．しかし，これは，複数の計算経路を正しく区別することができないため，各計算経路内で行われる lub 演算に基づく推論が正確にできない．より詳細にいうと，抽象領域の元は実領域の要素を多重に含んだものであり，projection transformer では，たまたま出力が同じ抽象領域の値になった場合に二つの計算経路を区別することができない．(順方向型の解析においても，同様なことが生じる)．たとえば，Even/odd 解析において，$f(x,y)$ という関数が，$(x,y)=(3,1)$ および $(1,\perp)$ により計算結果が2であるようなとき，$f(:\text{odd}, :\text{odd})=f(:\text{odd}, :\text{delayed})=:\text{even}$ であるが，(:odd, :delayed) のみが計算経路であるのか，(:odd, :odd) および (:odd, :delayed) の両方が計算経路となるのか区別がつかない．

そのため，projection transformer においては，if 文とは別に，特別なリスト操作に限定された条件文として case 文を導入している．そして，ストリクト性が正しく検出されるのは，case 文を用いた自己再帰関数の記述に限られる．実際，同じ宣言的意味を持つ関数を case 文と if 文を用いた異なる記述では，より一般的な if 文で記述されたプログラムに対する解析力は著しく制限される．

6.3 今後の課題

今後の課題としては，(1)対象となる性質の宣言的記述法およびその記述に基づく抽象実行帰納法の実行系の自動生成，(2)検証の対賀となる性質の拡張，がある．

(1)の宣言的記述については，4.1 節の種々のストリクトネスの定義のように，＋や×，およびデータ構成子，分解子を用いた再帰方程式による性質の記述を考えたい．それらから，定理証明系を用いた抽象領域の自動生成をめざす．

(2)については，Alternating Bit Protocol などのストリームを用いた関数型言語で記述されるプロトコル[6] に対する，抽象実行帰納法による検証の可能性について考えたい．この検証は，等式検証に帰着される．一般には，抽象領域上の等式は実領域上の等式に持ち上げることはできないが，if 文の分岐のみに注目すると then 部，else 部ともに真に等しい値を伝搬している．アプローチとしては，計算経路解析 (CPA) で求めた各計算経路上を "x が nil でないリストのとき，cons (car(x), cdr(x))$\Rightarrow x$" などの記号実行によるローカルな推論により等式関係を追跡する手法などを考えたい．

また，本手法の応用として，より精度の高い変則性検出[12] や，プログラム変換/部分計算[15] に対する制御なども考えてみたい．

7. おわりに

本稿では，関数型プログラムの帰納的性質を求める新たな手法として，抽象実行帰納法を提案した．この手法は，検証したい性質から抽象領域を定め，自動生成系により広域解析プログラムを生成する．そして，解析対象プログラムの抽象領域上の最上不動点を求め，

その性質を検証する．本手法は，検証系の停止性が保証されるので全自動の機械的検証が可能なほか，検証系の自動生成が可能という特徴を持つ．

謝辞 日頃，ご指導頂く NTT 基礎研究所情報科学研究部およびソフトウェア基礎研究所技術部の皆様に感謝します．

参考文献

1) Abramsky, S. and Hankin, C. (eds.): *Abstract Interpretation of Declarative Languages*, Ellis Horwood Limited (1987).
2) Boyer, R.S. and Moore, J.S.: *A Computational Logic*, Academic Press (1979).
3) Loeckx, J. and Sieber, K.: *The Foundations of Program Verification*, 2nd ed., John Wiley & Sons (1987).
4) Manna, Z.: *Mathematical Theory of Computation*, McGraw-Hill (1974).
5) Burstall, R.M. and Darlington, J.: A Transformation System for Developing Recursive Programs, *J. ACM*, Vol. 24, No. 1, pp. 44-67 (1977).
6) Dybjer, P. and Sander, H.P.: A Functional Programming Approach to the Specification and Verification of Concurrent Systems, *Formal Aspects of Computing*, Vol. 1, pp. 303-319 (1989).
7) Fosdick, L.D. and Osterweil, L.J.: Data Flow Anomaly Detection, *IEEE Trans. Softw. Eng.*, Vol. SE-10, No. 4, pp. 432-437 (1984).
8) Olender, K.M. and Osterweil, L.J.: Cecil: A Sequencing Constraint Language for Automatic Static Analysis Generation, *IEEE Trans. Softw. Eng.*, Vol. SE-16, No. 3, pp. 268-280 (1990).
9) Burn, G.L.: A Relationship between Abstract Interpretation and Projection Analysis (Extended Abstrct), *Proc. 17th ACM Sympo. on Principle of Programming Languages*, pp. 151-156 (1990).
10) Musser, D.L.: On Proving Inductive Properties of Abstract Data Types, *Proc. 7th ACM Sympo. on Principle of Programming Languages*, pp. 154-162 (1980).
11) Wadler, P. and Hughes, J.: Projections for Strictness Analysis, *Proc. Functional Programming Languages and Computer Architecture*, LNCS 274, pp. 385-407, Springer-Verlag (1987).
12) Ogawa, M. and Ono, S: Transformation of Strictness-related Analyses Based on Abstract Interpretation, 信学論, J74-E, No. 2, pp. 406-416 (1991).
13) Ogawa, M. and Ono, S.: Detecting Non-monotonic Properties in Functional Programs, 信学技報, SS 90-107, pp. 1-7, 信学会コンピューテーション研究会 (1990).
14) 小川瑞史，小野 諭：広域データフロー解析に基づく関数型プログラムの変則性検出，信学論，J71-D, No. 10, pp. 1949-1958 (1988).
15) Ono, S.: Relationships among Strictness-related Analyses for Applicative Languages, *Programming of Future Generation Computers II*, pp. 257-283, North-Holland (1988).
16) Ono, S., Ogawa, M. and Tsuruoka, Y.: Computation Path Analysis with Path Valid Condition, 信学技報, SS 89-16, pp. 41-50, 信学会ソフトウェアサイエンス研究会 (1989).
17) 小野 諭，小川瑞史，鶴岡行雄：最小不動点計算に基づくプログラムの帰納的性質の導出—広域解析自動生成系によるアプローチ，情報処理学会並列処理シンポジウム JSPP '90 論文集, pp. 249-256 (1990).

(平成2年8月27日受付)
(平成3年4月 9 日採録)

小川 瑞史（正会員）

昭和35年生．昭和58年東京大学理学部数学卒業．昭和60年同大大学院修士課程修了．同年日本電信電話(株)武蔵野電気通信研究所入所．以来，関数型プログラムの解析・検証，項書き換え系の研究に従事．現在，NTT 基礎研究所研究主任．電子情報通信学会，ACM 各会員．

小野 諭（正会員）

昭和29年生．昭和52年東京大学工学部電子卒業．昭和57年同大大学院博士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．以来，並列計算機，並列プログラム，ソフトウェア開発環境の研究に従事．現在，NTT ソフトウェア研究所主幹研究員．工学博士．電子情報通信学会，ACM，IEEE 各会員．