関数型プログラムの不変条件のICE流学習手法

千葉 知也　佐藤 亮介　松田 一孝　小林 直樹

プログラムの不変条件の発見はプログラム検証での鍵となるステップである．Garg らは機械学習によって不変条件を発見する，ICE と呼ばれる手法を提案した．ICE は，学習用データとして，通常の正例，負例に加え含意制約というものを含めることにより，帰納的な不変条件を効果的に発見できる．本研究では，ICE を一階の関数型プログラムの不変条件を発見出来るように拡張する．そのため，含意制約や負例を，複数回関数呼び出しする相互再帰関数の不変条件を発見するのに適する形に拡張し，それに応じて不変条件の学習法も拡張する．さらに，教師が学習者の推測を検証し，間違っているならば新たな学習用のデータを発見するアルゴリズムを与える．この手法を実装し，実験により有効性を確認した．

1　はじめに

不変条件とは，プログラムの実行中に常に満たされる条件であり，その発見はプログラム検証において鍵となるステップである．実際，高階モデル検査に基づいた関数型プログラム用自動検証器である MoCHi [5,6,9] のボトルネックとなっているのは，プログラムの抽象解釈やプログラムを述語抽象するのに適切な述語を発見することである．

本研究の目的は関数型プログラムの不変条件の発見である．ここでの関数型プログラムにおいての不変条件とは，各関数の事前条件と事後条件，すなわちプログラムの実行中の各関数呼び出しの引数と返り値の間に常に満たされる条件を指すものとする．例えば，図 1 に示すマッカーシーの９１関数を計算するプログラムの場合，引数 x の値は 111 以下なので事前条件は $x \leq 111$，x の値が 101 以下ならば返り値 r は 91 なので事後条件は $x \leq 101 \Rightarrow r = 91$ である．このような不変条件を発見するのが本研究の目的である．

```
let rec mc91 x =
  if x > 100 then
    x - 10
  else
    mc91 (mc91 (x+11))
in
  if n < 102 then assert (mc91 n = 91)
```

図 1　関数型プログラムの例

本研究では，Garg らによる不変条件発見手法 ICE [4] を関数型プログラム用に拡張する．ICE は機械学習手法の一種であり，不変条件を推測する学習者と，その不変条件の妥当性を検証する教師の 2 種類のモジュールから構成される．学習者はプログラムの内容を知らず，教師から与えられる学習データのみから不変条件の候補を見つけ，教師は，プログラムと照らしあわせてその候補が妥当であるか否かを判定し，妥当でなければその候補に矛盾する学習データを学習者に伝える，というやりとりを繰り返す．

通常の教師つき機械学習と比較して ICE の特徴的な点は学習データにある．ICE の学習データは，不

ICE-based Learning of Invariants for Functional Programs

Tomoya Chiba, Ryosuke Sato, Naoki Kobayashi, 東京大学情報理工学系研究科, Graduate School of Information Science and Technology, The University of Tokyo.

Kazutaka Matsuda, 東北大学大学院情報科学研究科, Graduate School of Information Science, TOHOKU University.

変条件 $I(x)$ を満たすべき正例 x の (部分) 集合，不変条件を満たすべきではない負例 y の集合に加えて，「$I(x)$ が成り立つならば $I(y)$ も成り立つ」ことを表す**含意制約** (x, y) の集合からなる．含意制約が必要であるのは，教師による不変条件の妥当性判定のために，プログラムから生成される検証条件が用いられるからである．手続き型言語におけるループ不変条件に関する検証条件式は $I(x) \wedge Q(x, y) \Rightarrow I(y)$ の形をしているため，学習が成功するためには，不変条件の中でも，特にそのような条件式を満たす不変条件 I（このような不変条件は一般に**帰納的**不変条件と呼ばれる）が見つかる必要がある．したがって，学習者にも $I(x) \wedge Q(x, y) \Rightarrow I(y)$ を満たす x, y の組を含意制約として与えなければ効果的に不変条件の発見を行うことはできない．

本論文では，上記の ICE の含意制約を，「$I(x_1), \ldots, I(x_n)$ が成り立つならば $I(y)$ も成り立つ」ことを表す $(\{x_1, \ldots, x_n\}, y)$ の形に拡張する．これは，多重再帰を含む関数型プログラムの場合，検証条件式が $I(x_1) \wedge \cdots \wedge I(x_n) \wedge P(x_1, \ldots, x_n) \Rightarrow I(y)$ の形になるからである．例えば，図 1 のプログラムの場合，`mc91` の事後条件を $I(x, r)$，事前条件を簡単のために `true`（恒真）とすると，else 部に関する検証条件式は

$$I(x + 11, r') \wedge I(r', r) \wedge \neg(x > 100) \Rightarrow I(x, r)$$

である．したがって，この場合には

$$(\{(x_1, r_1), (x_2, r_2)\}, (x, r))$$

の形の含意制約が必要である．

本論文では，相互再帰や多重再帰を許す一階の関数型言語を対象とし，上記のような含意制約の拡張を施した上で，学習者および教師の手続きを定式化する．また，実装・実験によりその有効性を確認する．本研究の貢献は以下の通りである．

- 多重再帰や相互再帰を許す一階の関数型プログラムの不変条件を発見できるように，ICE の学習データを拡張した．
- 学習データの拡張に合わせ，教師と学習者のアルゴリズムを拡張した．
- 実装および実験により，提案手法の有効性を確認した．

本論文の構成は以下の通りである．2 節では対象言語を定義する．3 節では我々の不変条件発見手法の定式化を行う．4 節では実装と実験結果について述べる．5 節では関連研究について議論し，6 節で結論と今後の課題を述べる．

2 対象言語

本節では，関数型プログラムの不変条件の発見の対象となる言語を導入する．

対象言語は，以下の構文で定義される一階の関数型言語である．

$$\begin{aligned} P &::= \text{let rec } D \text{ in } t \\ D &::= \{f_1(\widetilde{z_1}) = t_1, \ldots, f_n(\widetilde{z_n}) = t_n\} \\ t &::= x \mid \text{let } x = e \text{ in } t \mid \text{if } v \text{ then } t_1 \text{ else } t_2 \mid \mathit{fail} \\ e &::= v \mid \mathit{op}(v_1, v_2) \mid f(\widetilde{v}) \\ v &::= n \mid x \end{aligned}$$

メタ変数 P，D，t，e，v，n，x，op はそれぞれプログラム，関数定義の集合，項，単純式，値，整数，変数，整数上のプリミティブ演算を表す．$\widetilde{z}$ および $\widetilde{v}$ はそれぞれ変数および値の列を表す．関数 f_k の定義が $f_k(\widetilde{z}) = t$ のとき，$\widetilde{z}$ の長さを f_k のアリティと呼び，$\mathsf{arity}(f_k)$ と書く．算術式の定義で，$f(\widetilde{v})$ 中の $\widetilde{v}$ の長さは f のアリティと一致していなければならない．また，束縛変数名は互いに異なるものとする．ブール値は整数を使って表し，0 を false とみなす．fail はプログラムの異常終了を表す特別な項である．これを用いることで，表明 $\mathit{assert}(e)$ (e が true に評価されるべきであるという表明) は $\text{let } x = e \text{ in if } x \text{ then } \mathit{true} \text{ else } \mathit{fail}$ と書くことができる．簡単のため，この言語では A 正規化されたプログラムしか許していないが，一般のプログラムを A 正規化することができる [3] ため，これは本質的な制限ではない．

関数定義 D における簡約関係 $\longrightarrow_D$ は図 2 によって定義される．$[\![op]\!]$ は op によって示される演算であり，例えば，$[\![+]\!](1, 2) = 3$ となる．関数定義 D が文脈から明らかなときは，$\longrightarrow_D$ を $\longrightarrow$ と書く．

本研究の目的は，プログラムの実行が途中で停止しない (fail に到達しない) ことを証明するのに充分強い不変条件，すなわち，各関数毎の事前条件，事後条件を発見することである．ここで，$P = \text{let rec } D \text{ in } t$ かつ

$$
\begin{aligned}
VC(A, v, B) &= A \Rightarrow [v/r]B \\
VC(A, \mathit{fail}, B) &= \neg A \\
VC(A, \text{if } v \text{ then } t_1 \text{ else } t_2, B) &= VC(A \wedge v \neq 0, t_1, B) \wedge VC(A \wedge v = 0, t_2, B) \\
VC(A, \text{let } x = v \text{ in } t, B) &= VC(A \wedge x = v, t, B) \\
VC(A, \text{let } x = op(v_1, v_2) \text{ in } t, B) &= VC(A \wedge x = op(v_1, v_2), t, B) \\
VC(A, \text{let } x = f_k(\widetilde{v}) \text{ in } t, B) &= A \Rightarrow ([\widetilde{v}/\widetilde{z_k}]H_k^{\mathsf{pre}} \wedge VC([\widetilde{v}/\widetilde{z_k}, x/r_k]H_k^{\mathsf{post}}, t, B))
\end{aligned}
$$

$$
VC_p(\text{let rec } \{f_1(\widetilde{z_1}) = t_1, \ldots, f_n(\widetilde{z_n}) = t_k\} \text{ in } t) = \left(\bigwedge_{k \in \{1,\ldots,n\}} VC(H_k^{\mathsf{pre}}, t_k, H_k^{\mathsf{post}}) \right) \wedge VC(\mathit{true}, t, \mathit{true})
$$

図 3　検証条件 VC_p の生成規則

$$
E[\text{let } x = v \text{ in } t] \longrightarrow_D E[[v/x]t]
$$

$$
E[op(v_1, v_2)] \longrightarrow_D E[[\![op]\!](v_1, v_2)]
$$

$$
\frac{v \neq 0}{E[\text{if } v \text{ then } t_1 \text{ else } t_2] \longrightarrow_D E[t_1]}
$$

$$
\frac{v = 0}{E[\text{if } v \text{ then } t_1 \text{ else } t_2] \longrightarrow_D E[t_2]}
$$

$$
\frac{f(\widetilde{z}) = t \in D}{E[f(\widetilde{v})] \longrightarrow_D E[[\widetilde{z}/\widetilde{v}]t]}
$$

$$
E ::= [\,] \mid \text{let } x = E \text{ in } t
$$

図 2　意味論

$f(\widetilde{z}) = t' \in D$ のとき，P において H^{pre} が f の**事前条件**であるとは，任意の $\widetilde{v}$ について，$t \longrightarrow_D^* E[f(\widetilde{v})]$ のとき，$[\widetilde{v}/\widetilde{z}]H^{\mathsf{pre}}$ が成り立つことを言う．また，P において H^{post} が f の**事後条件**であるとは，任意の $\widetilde{v}$，v について，$t \longrightarrow_D^* E[f(\widetilde{v})]$ かつ $f(\widetilde{v}) \longrightarrow_D^* v$ のとき，$[\widetilde{v}/\widetilde{z}, v/r]H^{\mathsf{post}}$ が成り立つことを言う．以下，関数 f_k の事前条件，事後条件をそれぞれ H_k^{pre}，H_k^{post} と書く．

事前条件と事後条件の候補が与えられたとき，それらが実際に事前条件，事後条件になっていること，プログラム P の実行が途中で異常終了しないことの検証条件 $VC_p(P)$ は，図 3 のように定義できる．ここで，$VC(A, t, B)$ は，A が成り立つときに t の評価結果 r が B を満たすための十分条件を表す．$VC_p(\text{let rec } D \text{ in } t)$ が恒真のとき，$t \not\longrightarrow_D^* E[\mathit{fail}]$ が成り立つ．

3　発見手法

本節では，本研究の提案である，関数型言語のプログラムの不変条件手法について述べる．まず，学習データの拡張について述べ，その後，教師と学習者の具体的なアルゴリズムについて述べる．

3.1　拡張された学習データ

学習データは $\{\xi_1, \ldots, \xi_n\}$ という形の**負例制約**の集合 C と，$(\{\xi_1, \ldots, \xi_n\}, \xi')$ という形の**含意制約**の集合 I から構成される．1 節にて，含意制約を拡張する必要があることは既に述べた．負例制約を拡張するのも，同様に多重再帰を扱うためである．ここで，各 $\xi_1, \ldots, \xi_n$ および ξ は，**データポイント**であり，3 つ組 $(\mathsf{pre}, k, \widetilde{v})$ もしくは 4 つ組 $(\mathsf{post}, k, \widetilde{v}, a)$ のいずれかの形をしている．データポイント ξ が**例**であるとは，もし $\xi = (\mathsf{pre}, k, \widetilde{v})$ という形ならば，求める関数 f_k の事前条件が引数 $\widetilde{v}$ に対し成り立つこと，もし $\xi = (\mathsf{post}, k, \widetilde{v}, a)$ という形ならば，求める関数 f_k の事後条件が引数 $\widetilde{v}$ とそのときの返り値 a に対し成り立つことと定義する．

直感的には，負例制約 $\{\xi_1, \ldots, \xi_n\}$ は，少なくとも一つの ξ_i が例ではない（すなわち，負例である）という制約を表し，含意制約 $(\{\xi_1, \ldots, \xi_n\}, \xi')$ は，もし $\{\xi_1, \ldots, \xi_n\}$ が全て例であれば ξ' もまた例でなけ

ればならないという制約を表す.

元の ICE とは異なり，我々の拡張において，正例（制約）は，含意制約の $(\emptyset, \xi)$ という形の特殊な場合となる.

3.2 教師アルゴリズム

次に教師アルゴリズムについて述べる．教師は，学習から事前・事後条件の候補を受けとり，その候補の妥当性をプログラムと照らしあわせて検査し，もし妥当でなければ新たな学習データを生成し学習者に渡す．教師アルゴリズムは以下の 3 ステップからなる.

1. 対象プログラムと，学習者より受けとった各関数毎の事前条件の候補 H_k^{pre} と事後条件の候補 H_k^{post} から図 3 に従い検証条件 VC_p を生成する.
2. SMT ソルバを用いて，検証条件が恒真かどうか検査する.
3. もし恒真であれば，その検証条件を出力し，拡張 ICE を終了する.

3' もし恒真でなければ，SMT ソルバから得られた反例から新たな学習データを作成し，学習者に渡す.

3.2.1 学習データの生成

ステップ 3' における学習データの生成について詳しく述べる．VC_p が恒真でないため，$VC_p\theta$ が偽となるような代入 θ が存在することなる．一方で，VC_p は以下のいずれかの形をしたホーン節の連言で表現できる.

$$(P_1 \wedge \cdots \wedge P_n \wedge C) \Rightarrow P' \qquad \text{(H1)}$$

$$(P_1 \wedge \cdots \wedge P_n \wedge C) \Rightarrow C' \qquad \text{(H2)}$$

ここで，$P_1, \ldots, P_n$ および P' は $[\widetilde{x_k}/\widetilde{z_k}]H_k^{\mathsf{pre}}$ もしくは $[\widetilde{x_k}/\widetilde{z_k}, y/r_k]H_k^{\mathsf{post}}$ のいずれかの形の論理式であり，C はそれ以外の形の論理式である．今，θ の下で VC_p が偽であるので，こうした形のホーン節の中に θ の下で偽になるものが少なくとも一つ存在することになる.

もし，θ の下で (H1) の形のホーン節が偽であったとする．この式を真にするためには，$P_1, \ldots, P_n$ のいずれかが偽であるか，P' が真であることが必要である．このとき，それに対応して，教師は含意制約 $(\{\eta(P_1), \ldots, \eta(P_n)\}, \eta(P'))$ を生成する．ただし，$\eta(P)$ は以下で定義される関数である.

$$\eta(P) = \begin{cases} (\mathsf{pre}, k, \theta(\widetilde{x})) & \text{if } P = [\widetilde{x_k}/\widetilde{z_k}]H_k^{\mathsf{pre}} \\ (\mathsf{post}, k, \theta(\widetilde{x}), \theta(y)) & \text{if } P = [\widetilde{x_k}/\widetilde{z_k}, y/r_k]H_k^{\mathsf{post}} \end{cases}$$

一方，θ の下で (H2) の形のホーン節が偽であったとする．この式を真にするためには，$P_1, \ldots, P_n$ のいずれかが偽であることが必要である．このとき，それに対応して，教師は負例制約 $\{\eta(P_1), \ldots, \eta(P_2)\}$ を生成する.

教師は，こうして生成された含意制約と負例制約を集めて，以前の教師フェイズまでで得られた含意制約と負例制約に追加し，学習者に渡す.

3.3 学習者アルゴリズム

最後に，学習者アルゴリズムについて説明する．学習者は，教師から学習データ I, C を受けとり，それのみから各関数の事前・事後条件の候補を発見し，それらを教師へと渡す.

学習者のアルゴリズムは以下の 4 ステップからなる.

1. 各関数について，事前・事後条件のテンプレートを準備する.
2. テンプレートのパラメータに関する制約を，教師から受けとった学習データより求める.
3. 制約を SMT ソルバ等を利用して解消し，具体的な事前・事後条件の候補を求める.
4. 得られた各関数毎の事前・事後条件の候補を教師に渡す.

以下，ステップ 1 で準備するテンプレートと，ステップ 2 で生成する制約について詳しく述べる.

3.3.1 事前・事後条件のテンプレート

事前・事後条件のテンプレートには，ICE [4] と同様に，以下の形に表される，線形不等式の L_j 個の論理積の L_i 個の論理和を用いる.

$$\hat{H}_k^t(x_1^k, ..., x_m^k) = \bigvee_{i \in \{1,\ldots,L_i\}} \bigwedge_{j \in \{1,\ldots,L_j\}} (s_{kt1}^{ij} x_{v_{kt1}^{ij}}^k + s_{kt2}^{ij} x_{v_{kt2}^{ij}}^k \leq c_{kt}^{ij})$$

ここで，t はシンボル pre もしくは post であり，k は関数 f_k に対するテンプレートであることを表す．また，$x_1^k, ..., x_m^k$ は f_k の引数と返り値を表す変数であ

る．つまり，$t = \mathsf{pre}$ のときは $(x_1^k, ..., x_m^k) = (\widetilde{z_k})$ であり，$t = \mathsf{post}$ のときは $(x_1^k, ..., x_m^k) = (\widetilde{z_k}, r_k)$ である．

上記テンプレートにおいて L_i と L_j は，初めは小さな値（例えば 1 や 2）にしておき，解が見つからなかったときに徐々に大きくすればよい．(ただし，4 節で述べる実装では，簡単のため，プログラムごとに固定している．) 一般に L_i や L_j が大きくなると制約解消の時間が大きく増加する．一方，L_i や L_j が小さすぎると適切な事前・事後条件の候補がテンプレートで表現できない場合がある．

3.3.2 テンプレートのパラメータに関する制約の生成と解消

学習は，教師から受けとった学習データ C，I から，テンプレートのパラメータ s_{ktl}^{ij}，v_{ktl}^{ij}，c_{kt}^{ij} に関する制約 ψ を図 4 に従い生成する．この制約 ψ の構成は，元の ICE の構成法の単純な拡張になっている．ここで，i，j，k，t，l および x の動く範囲は，i が $\{1, \ldots, L_i\}$，j が $\{1, \ldots, L_j\}$，k は対象プログラムの関数の集合を $\{f_1, \ldots, f_n\}$ とすると $\{1, \ldots, n\}$，t は $\{\mathsf{pre}, \mathsf{post}\}$，$l$ は $\{1, 2\}$，そして x は学習データ I と C 中のデータポイント全体である．制約 ψ は，s_{ktl}^{ij}，v_{ktl}^{ij}，c_{kt}^{ij} 以外に，変数 b_x，b_x^{ij}，d_{xl}^{ij} および r_{xl}^{ij} を含んでいる．これらの変数の直感的意味は以下の通りである．

- b_ξ は，ξ が例であるかどうかを表す真偽値．
- b_ξ^{ij} は，ξ に対する (i, j) 番目の線形不等式の真偽値．すなわち，$s_{kt1}^{ij} x_{v_{ktl1}^{ij}} + s_{kt2}^{ij} x_{v_{kt1}^{ij}} \leq c_{kt}^{ij}$ の，$k = K(\xi)$, $t = T(\xi)$，$x_{v_{kt2}^{ij}} = \mathsf{nth}(\xi, v_{kt2}^{ij})$ としたときの真偽値．
- $d_{\xi l}^{ij}$ は，ξ に対する $s_{kt2}^{ij} x_{v_{ktl}^{ij}}$ の値．
- $r_{\xi l}^{ij}$ は，ξ に対する $x_{v_{ktl}^{ij}}$ の値．

また，M は定数であり，L_i や L_j と同様に，小さな値からはじめて解が見つからない場合に徐々に大きくしていく．

一見，変数 b_ξ，b_ξ^{ij}，$d_{\xi l}^{ij}$ および $r_{\xi l}^{ij}$ は不要に見えるかもしれない．しかし，これらの変数を利用することにより，ψ は量化子のない一階の整数線形算術の論理式として表現でき [4]，その充足可能性の判定を，Z3 [2] などの SMT ソルバを用いて行うことができる．

テンプレートのパラメータに関する制約 ψ が得られた後，学習者は SMT ソルバ等を利用することで制約 ψ を解消する．もし制約解消が成功し，$\psi\theta$ が真となるような代入 θ を得られれば，事前・事後条件の候補 H_k^t を $\hat{H}_k^t\theta$ により得て，これを教師に渡す．

もし，そのような代入が得らない場合は，現在のテンプレートでは事前・事後条件が表現できないことを意味する．その場合は定数 L_i，L_j，M を増加して再び拡張 ICE を実行する．

4 実装と実験

提案手法に基づき，不変条件発見器のプロトタイプを OCaml で実装し，実験による評価を行った．

4.1 実装

不変条件発見は 3 節で述べた手法を用い，そのうち，学習者の不変条件候補の制約式と，教師の不変条件が検証条件を満たすかを表す式を解くための SMT ソルバとして Z3 [2] を用いた．不変条件候補の制約式は，制約がインクリメンタルに増えていくため，Z3 のインクリメンタルに制約を解く機能を用いた．

事前条件のテンプレートを論理和と論理積の数は 1 に固定し，事後条件の論理和と論理積の数および各テンプレートの定数項の範囲は対象プログラムごとにユーザが指定するようにした．

4.2 実験

複数の簡単なプログラムについて，実験を行った結果を図 5 に示す．

図 5 中の L_i は，3.3.1 節の事後条件のテンプレート中の節の論理和の個数であり，L_j は各節中の論理積の個数である．M は，テンプレート中の線形不等式の上限の範囲であり，各 c_{kt}^{ij} は，$-M \leq c_{kt}^{ij} \leq M$ を満たす．また，図 5 中の「学習データの個数」は，不変条件が見つかった時点での学習データの個数である．

実験に用いたプログラムは以下のとおり．

- ack，sum，mc91 は，MoCHi [5] のベンチマーク集に含まれるものである．mc91 はマッカーシーの 9 1 関数を評価して，引数が 101 以下な

$$
\begin{aligned}
&\psi\left((s^{ij}_{ktl})_{i,j,k,t,l},(v^{ij}_{ktl})_{i,j,k,t,l},(c^{ij}_{kt})_{i,j,k,t}\right) := \\
&\quad\left(\bigwedge_{\Xi\in C}\left(\bigvee_{\xi\in\Xi}\neg b_\xi\right)\right)\wedge\left(\bigwedge_{(\Xi,\xi')\in I}\left(\bigwedge_{\xi\in\Xi}b_\xi\right)\Rightarrow b_{\xi'}\right)\wedge\left(\bigwedge_{\xi}\left(b_\xi\Leftrightarrow\bigvee_i\bigwedge_j b^{ij}_\xi\right)\right)\\
&\quad\wedge\left(\bigwedge_{\xi,i,j}\left(b^{ij}_\xi\Leftrightarrow\left(\sum_{l\in\{1,2\}}r^{ij}_{\xi l}\le c^{ij}_{K(\xi)T(\xi)}\right)\right)\right)\wedge\left(\bigwedge_{i,j,k,t}\left(-M\le c^{ij}_{kt}\le M\right)\right)\\
&\quad\wedge\left(\bigwedge_{\substack{\xi,i,j\\ l\in\{1,2\}}}\left(\begin{array}{l}s^{ij}_{K(\xi)T(\xi)l}=0\Rightarrow r^{ij}_{\xi l}=0\\ s^{ij}_{K(\xi)T(\xi)l}=1\Rightarrow r^{ij}_{\xi l}=d^{ij}_{\xi l}\\ s^{ij}_{K(\xi)T(\xi)l}=-1\Rightarrow r^{ij}_{\xi l}=-d^{ij}_{\xi l}\end{array}\right)\right)\\
&\quad\wedge\left(\bigwedge_{\substack{\xi,i,j\\ l\in\{1,2\}}}\bigwedge_{h\in\{1,\ldots,\mathsf{arity}(K(\xi))+1\}}\left(v^{ij}_{K(\xi)T(\xi)l}=h\Rightarrow d^{ij}_{\xi l}=\mathsf{nth}(\xi,h)\right)\right)\\
&\quad\wedge\left(\bigwedge_{\substack{i,j,k,t\\ l\in\{1,2\}}}\left(-1\le s^{ij}_{ktl}\le 1\right)\right)\\
&\quad\wedge\left(\bigwedge_{\substack{i,j,k\\ l\in\{1,2\}}}\left(1\le v^{ij}_{k\,\mathsf{pre}\,l}\le\mathsf{arity}(f_k)\right)\wedge\bigwedge_{\substack{i,j,k\\ l\in\{1,2\}}}\left(1\le v^{ij}_{k\,\mathsf{post}\,l}\le\mathsf{arity}(f_k)+1\right)\right)\wedge\left(\bigwedge_{i,j,k,t}\left(v^{ij}_{kt1}\neq v^{ij}_{kt2}\right)\right)
\end{aligned}
$$

$$
\begin{aligned}
K((\mathsf{pre},k,\widetilde{v})) &:= k & \mathsf{nth}((\mathsf{pre},k,\widetilde{v}),h) &:= v_h\\
K((\mathsf{post},k,\widetilde{v},a)) &:= k & \mathsf{nth}((\mathsf{post},k,\widetilde{v},a),h) &:= \begin{cases}v_h & \text{if } 1\le h\le\mathsf{arity}(f_k)\\ a & \text{if } h=\mathsf{arity}(f_k)+1\end{cases}\\
T((\mathsf{pre},k,\widetilde{v})) &:= \mathsf{pre}\\
T((\mathsf{post},k,\widetilde{v},a)) &:= \mathsf{post}
\end{aligned}
$$

図 4　テンプレートのパラメータに関する制約式

Program	L_i	L_j	M	学習データの個数	時間
ack	1	1	10	21	0.59
sum	2	2	30	4	0.12
mc91	2	2	110	135	975.49
mc11	2	2	30	135	26.55
mc11-multi	2	2	30	373	697.72
fib	2	2	10	30	3.68
mutual1	2	2	10	101	25.53
mutual2	2	2	30	108	29.95

図 5　実験結果

ら返り値が 91 であることをチェックし，成り立たなければ *fail* するプログラムである．

- mc11 は mc91 の変種であり，引数が 21 以下なら返り値が 11 になることをチェックするプログラムである．mc11-multi は mc11 を相互再帰を用いるように変形したものである．
- fib は，n 番目のフィボナッチ数列を再帰を用いて計算し，結果が $n-1$ 以上であることをチェッ

クするプログラムである．

- mutual1, mutual2 は相互再帰を用いたプログラムである．

いずれのプログラムについても，プログラムが *fail* で異常終了しないことを検証するのに充分強い不変条件が求まった．プログラム mc91 に時間がかかっているのは，不変条件中に必要な定数が大きいためであり，そのような不変条件をより高速に求める方法を考えることは今後の課題である．また，mc11 に比べて mc11-multi の検証に時間がかかっているのは相互再帰のために求める不変条件の個数が倍になったためであると考えられる．また，実験全体を通して，学習データとして引数と返り値が 1 しか違わないなど，非常に似ているものが多く発見されており，より効果的な学習データの選別も今後の課題である．

5 関連研究

1 節で述べたとおり，本論文の手法は，Garg らによる不変条件発見手法 ICE [4] を関数型プログラム用に拡張したものである．ICE の含意制約は多重再帰を持つ関数の不変条件の発見には適していないため，含意制約を拡張した．

不変条件の発見手法の研究にはさまざまなものがあり，関数型プログラム用の不変条件発見手法にかぎっても，抽象解釈 [1] に基づく手法，interpolant に基づく手法 [7,8]，テスト実行に基づく手法 [10] など様々なものが提案されている．それらの中でも本論文の手法に関連が深いのは，関数型プログラム用の不変条件発見問題をホーン節集合に対する求解問題に帰着して解く手法 [8] である．ホーン節集合が本論文で用いた検証条件に対応しており，本論文の手法では学習者にはホーン節は示さずに例のみを与えているのに対し，上記手法 [8] ではホーン節集合の解を直接求めようとしている点に違いがある．

6 まとめと今後の課題

本論文では，不変条件学習手法 ICE の枠組みを拡張し，関数型プログラムの不変条件の発見手法を提案した．今後の課題としては，実装の改良，高階関数を扱うための拡張，他の関数型プログラムの不変条件発見手法との実験的比較などが挙げられる．

謝辞

本研究は JSPS 研究費 15H05706，23220001 の助成を受けたものです．

参 考 文 献

[1] Cousot, P. and Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, New York, NY, USA, ACM, 1977, pp. 238–252.

[2] de Moura, L. M. and Bjørner, N.: Z3: An Efficient SMT Solver, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, Ramakrishnan, C. R. and Rehof, J.(eds.), Lecture Notes in Computer Science, Vol. 4963, Springer, 2008, pp. 337–340.

[3] Flanagan, C., Sabry, A., Duba, B. F., and Felleisen, M.: The Essence of Compiling with Continuations, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, Cartwright, R.(ed.), ACM, 1993, pp. 237–247.

[4] Garg, P., Löding, C., Madhusudan, P., and Neider, D.: ICE: A Robust Framework for Learning Invariants, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, Biere, A. and Bloem, R.(eds.), Lecture Notes in Computer Science, Vol. 8559, Springer, 2014, pp. 69–87.

[5] Kobayashi, N., Sato, R., and Unno, H.: Predicate abstraction and CEGAR for higher-order model checking, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, Hall, M. W. and Padua, D. A.(eds.), ACM, 2011, pp. 222–233.

[6] Sato, R., Unno, H., and Kobayashi, N.: Towards a scalable software model checker for higher-order programs, *Proceedings of the ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation, PEPM 2013, Rome, Italy, January 21-22, 2013*, Albert, E. and Mu, S.(eds.), ACM, 2013, pp. 53–62.

[7] Terauchi, T.: Dependent types from counterexamples, *Proceedings of the 37th ACM SIGPLAN-*

SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010, Hermenegildo, M. V. and Palsberg, J.(eds.), ACM, 2010, pp. 119–130.

[8] Unno, H. and Kobayashi, N.: Dependent type inference with interpolants, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, Porto, A. and López-Fraguas, F. J.(eds.), ACM, 2009, pp. 277–288.

[9] Unno, H., Terauchi, T., and Kobayashi, N.: Automating relatively complete verification of higher-order functional programs, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, Giacobazzi, R. and Cousot, R.(eds.), ACM, 2013, pp. 75–86.

[10] Zhu, H., Nori, A. V., and Jagannathan, S.: Dependent Array Type Inference from Tests, *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, D'Souza, D., Lal, A., and Larsen, K. G.(eds.), Lecture Notes in Computer Science, Vol. 8931, Springer, 2015, pp. 412–430.