プログラミング 7－6
（1996. 5. 31）

抽象解釈における Lazy な抽象領域の生成

小川 瑞史
NTT 基礎研究所
mizuhito@theory.brl.ntt.jp

小野 諭
NTT ソフトウェア研究所
ono@slab.ntt.jp

本稿では、逆方向抽象解釈の自動生成にむけて、フラットドメイン上の解析を再帰領域方程式で定義される代数型を持つノンフラットドメイン上の解析に拡張する手法をプロジェクション解析を例に示す。この手法は検出したい性質の仕様記述に従い、解析実行時に lazy に抽象領域を生成し、検出したい性質のみに注目した lazy な不動点計算を行なう。

Lazy Abstract Domain Creation for Abstract Interpretation

Mizuhito Ogawa
NTT Basic Research labs.
mizuhito@theory.brl.ntt.jp

Ono Satoshi
NTT Software Labs.
ono@slab.ntt.jp

This paper presents the method to extend a backward analysis on a flat domain to those that on a nonflat domain specified by recursive domain equations. The running example is the *projection alalysis.* This method create the abstract domain lazily from the specification of an objetive property, which is called *Lazy Abstract Domain Creation.*

1 はじめに

近年、関数型言語における抽象解釈の手法は最適化コンパイル技術として広く研究されてきた [20]。従来の主たる抽象解釈の応用はストリクトネス解析であり、抽象解釈の研究は効率的なストリクトネス解析の実現法（解析力と効率的実現のトレードオフ [16, 12] や、抽象領域上での効率的な不動点計算 [3, 8, 9]）など実装レベルに近付きつつある。

最適化技術という観点を離れると、プログラムの性質を調べるという点で、解析と相補的なものとして検証がある。従来の検証法は非常に表現力の高い論理に基づき、強力な検証力を持つが、その代償に停止性が保証されず、適切な検証戦略のガイドが不可欠であった。この解析と検証の間を埋めるものとして、いくつかの関数型言語への制限のもとで、[11] では比較的易しい性質 ーたとえば sort が停止した場合の正当性ー について自動検証が可能であることを示した。

このような比較的易しい性質の自動検証を抽象解釈により行なう上で重要な点は

- 検証可能な性質の仕様記述言語
- 仕様記述された性質の解析系の自動生成

であろう。仕様記述に関しては単項二階論理 (monadic second order logic) [17] や集合制約 (set constraint) [2] などが親和性が高いと思われるが、まだ明らかではない。

解析の自動生成については、手続き型言語については Cecil [14] があり、関数型言語では [21, 19] などがあるが、前者は非常に限られた性質（正規表現で記述できる手順の検出）に制限され、後者は不動点計算と抽象解釈の仕様の分離に留まり、抽象領域や抽象解釈に必要な演算をすべて記述する必要があった。

また高階関数型言語のストリクトネス解析における抽象領域の自動生成は、[1] において提案されているが、それは抽象領域の構成のみを対象としたものであり、抽象解釈に必要な演算についてはふれていない。

本稿では、再帰領域方程式により代数的に定義される型をもつ一階の関数型言語を対象とし、解析対象となる性質の仕様をもとに逆方向の抽象解釈を自動生成する手法の概略について、プロジェクション解析の例に基づき説明する。

この自動生成法は、ベースとなる領域（たとえば、Bool 値とか整数値）において定義された解析を、その上の代数型に自動的に拡張する。その拡張は、代数型の定義に対応する再帰的スキーマにより解析対象の性質を仕様記述し、その仕様に基づき解析の実行時に動的に抽象領域および抽象解釈に必要な演算を lazy に生成することによりなされる。

本稿では章 2 で、代数型、抽象解釈、プロジェクション解析などの用語について説明する。章 3 で、lazy な抽象領域および抽象解釈に必要な演算の生成、およびそれにともない必要となる lazy な不動点計算について概略を説明する。章 4 で、プロジェクション解析を例として lazy な抽象領域および抽象解釈に必要な演算の生成を詳述する。

2 定義と準備

2.1 再帰的領域方程式と遅延代数型

本稿で対象とする言語は、一階の関数型言語で、以下の文法を満たす再帰方程式 E により代数的に $\mu X.E$（変数はすべて μ により束縛される）として定義される型（または Bool 型 $= \{true, false\}$）を持つとする。

$$E := X \mid c \mid f(E, \cdots, E) \mid E + E \mid E \times E$$

たとえば、整数型 Nat は $Nat := \mu X.0 + succ(X)$ により定義され、型 α に対するリスト型は $list(\alpha) := \mu X.nil + cons(\alpha, X)$ として定義される。（整数型 Nat は、説明の必要に応じて Bool 型のような built-in-type として扱うこともある。）

また、型の導入と同時に、構成子、選択子、識別子などの基本関数が導入される。それらはたとえば、

基本関数	構成子	選択子	識別子
Nat	$succ(x)$	$pred(x)$	$zerop(x)$
$list(\alpha)$	$cons(x, y)$	$head(x), tail(x)$	$null(x)$

などである。

2.2 逆方向の抽象解釈

抽象解釈は、再帰方程式で記述された関数型プログラムを注目する性質について抽象化した領域上で不動点計算により解釈する手法であり、広域解析などの形式化の手法としてしばしば用いられている [20]。通常、抽象領域は有限の領域であり、性質を近似的に解析する代償として不動点計算の停止性が保証される。

抽象解釈には大きく分けて、順方向実行と逆方向実行の二つがある [13, 10]。直観的には、順方向実行は入力値にある性質を仮定したとき出力値がどんな性質を持つか計算するものであり、逆方向実行は出力値がある性質を持つためには入力値がどんな性質を満たすことが必要であるか計算する。

たとえば、整数上で偶数・奇数を解析するための抽象領域は $\{even, odd, \bot, \#\}$ である（ただし $\bot \sqsubseteq even, odd \sqsubseteq \#$、$\#$ は矛盾を表す）が、if 文についてみると順方向実行では

$$\begin{aligned} if\ :\ & (true, even, \bot) \to even, \quad (true, odd, \bot) \to odd, \\ & (true, \bot, even) \to \bot, \quad (false, even, \bot) \to \bot, \\ & (false, \bot, even) \to even, \quad (false, \bot, odd) \to odd, \\ & \cdots \end{aligned}$$

などから、$(true, even, *), (false, *, even)$（ただし $*$ は $even, odd, \bot$ のいずれか）の時に、出力が $even$ となることを計算する。逆方向実行では

$$if\ :\ even \to \{(true, even, \bot), (false, \bot, even)\}$$

と計算する。本稿では逆方向実行のフレームワークを扱う。

逆方向の実行で重要な作用素は $\sqcup$ と $\&$ の二つである。$\sqcup$ は前掲の if 文の逆方向実行などで現われるように、可能性のある入力値の組が二つ以上あるとき、に用いる。前掲の例は $\sqcup$ を用いて、

$$if\ :\ even \to (true, even, \bot) \sqcup (false, \bot, even)$$

と表される。（以後、この表記を用いる。）$\&$ は、たとえば関数本体内で変数 x が複数箇所で用いられている時、それぞれが満たすべき性質をマージ指せるために用いる。たとえば、ある場所の x が $even$ であることを要求され、他の場所の x が odd であることを要求されると、$even \& odd = \#$ として矛盾が生じ、そのような入力値を求める計算は中止される。詳しくは [20] を参照されたい。

表 1: フラットドメインにおけるプロジェクションの例

Element	*Projection*			
	ID	*STR*	*ABS*	*FAIL*
$u\ (\neq \bot, \lightning)$	u	u	$\bot$	$\lightning$
$\bot$	$\bot$	$\lightning$	$\bot$	$\lightning$
$\lightning$	$\lightning$	$\lightning$	$\lightning$	$\lightning$

2.3 プロジェクション解析

本稿では逆方向実行の実例として、ストリクトネス解析の拡張であるプロジェクション解析を用いる [18, 7, 5]。遅延評価型の関数型言語において, $f(x_1, \cdots, x_n)$ で引数 x_i がストリクトであるとは, $f(x_1, \cdots, x_{i-1}, \bot, x_{i+1}, \cdots, x_n) = \bot$ がすべての $x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n$ について成り立つことをいう. これは, 引数 x_i の評価が $f(x_1, \cdots, x_n)$ の評価において必須であり, x_i の先評価という最適化がプログラムの意味を変えない安全な操作であることを意味する. 再帰的に定義された関数 f のストリクト性を検出することをストリクトネス解析という.

プロジェクション解析は、領域上の一引数関数（プロジェクション）の抽象化に基づく抽象実行であり、ストリクト性と同時に不要性（たとえばリスト上のヘッドストリクト性）も解析できる点が通常のストリクトネス解析の拡張となっている。

領域 D から誘導されるプロジェクション α とは, 以下の性質をもつ $D_\lightning \to D_\lightning$ の写像である. ただし $D_\lightning = D \cup \{\lightning\}$ かつ $\lightning$ は $D_\lightning$ の最小元とし[1]、関数 $f : D \to D$ は, $f(\lightning) = \lightning$ として, $D_\lightning \to D_\lightning$ に自然に拡張してあるものとする。

- α は連続（したがって単調）写像
- $\alpha \sqsubseteq ID_{D_\lightning}$
- $\alpha \circ \alpha = \alpha$ (idempotent).

フラットドメイン上の典型的なプロジェクション *ID*, *STR*, *ABS*, *FAIL* を表 1に示す.

これらのプロジェクションは, フラットな領域 A 上の線形リスト $list(A) = \mu X.nil + cons(A, X)$ において、A 上のプロジェクション α に対し,

$$FIN\,\alpha = \mu X.NIL \sqcup CONS(\alpha, X),$$

$$INF\,\alpha = \mu X.NIL \sqcup CONS(\alpha, X \sqcup ABS)$$

なるスキーマにより拡張される。たとえば、

$$HEAD = ABS \sqcup INF\ STR$$

$$TAIL = ABS \sqcup FIN\ ID$$

$$TOTAL = ABS \sqcup FIN\ STR$$

などである。ここで $\sqcup$ は if 文の条件分岐に対応し、$\alpha \sqsubseteq \beta \Leftrightarrow \forall x \in D_\lightning.\ \alpha(x) \sqsubseteq \beta(x)$ により定義される順序 $\sqsubseteq$ の lub になる。$\sqcup$ の満たす規則は、巾等則、交換則、結合則である。ノンフラットドメインについては、さらに構成子（たとえば *CONS*）に関する分配則が成り立つ（たとえば $CONS(\alpha \sqcup \beta, \gamma) = CONS(\alpha, \gamma) \sqcup CONS(\beta, \gamma)$）。たとえばフラットドメイン上のプロジェクション *ID*, *STR*, *ABS*, *FAIL* においては, その順序関係は図 1となる. (*ID* は *STR* $\sqcup$ *ABS* として, *FAIL* は *STR* & *ABS* として表される。) ノンフラットドメイン上のプロジェクション *HEAD*, *TAIL*, *TOTAL* などにおいては, その順序関係は図 2となる.

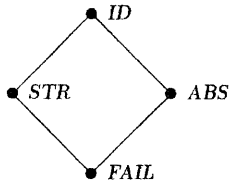

図 1: フラットドメイン上プロジェクションの順序

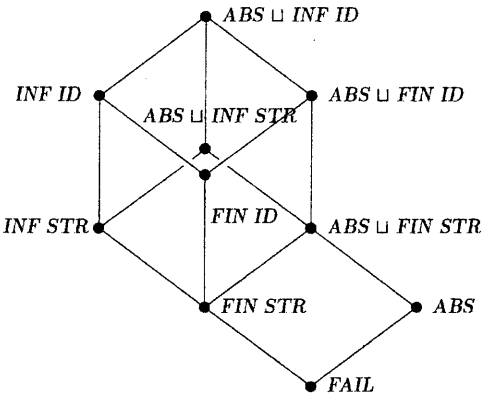

図 2: 線形リストのプロジェクションのなす抽象領域

& は同一変数の複数インスタンスへのデマンドを併合するオペレータで, 矛盾するデマンドの併合は *FAIL* となる. $\alpha\&\beta$ は, 以下の規則により定義される.

$$\alpha\,\&\,\alpha = \alpha \quad \text{(巾等則)}$$

$$\alpha\,\&\,\beta = \beta\,\&\,\alpha \quad \text{(交換則)}$$

$$\alpha\,\&\,(\beta\,\&\,\gamma) = (\alpha\,\&\,\beta)\,\&\,\gamma \quad \text{(結合則)}$$

$$\alpha\,\&\,(\beta \sqcup \gamma) = (\alpha\,\&\,\beta) \sqcup (\alpha\,\&\,\gamma) \quad (\sqcup \text{ 上の分配則})$$

$$ABS\,\&\,\alpha = \alpha$$

$$FAIL\,\&\,\alpha = FAIL$$

さらにノンフラットドメイン、たとえば線形リストの場合には以下の規則が加わる.

$$CONS(\alpha, \beta)\,\&\,NIL = FAIL$$

$$CONS(\alpha, \beta)\,\&\,CONS(\gamma, \delta) = CONS(\alpha\,\&\,\gamma, \beta\,\&\,\delta)$$

プロジェクション解析の解析対象は, $\alpha \circ f \sqsubseteq f \circ \beta$ という関係である. (この両辺に左から α を適用すると, $\alpha \circ f \sqsubseteq$

[1]直観的には, $\bot$ が「未評価」, 「情報の欠如」を表すのに対し, $\lightning$は, 「計算が無効であること」, 「矛盾していること」などを表す.

$\alpha \circ f \circ \beta$ となる.) たとえば, フラットドメイン上の関数 $f(x)$ において, ストリクトであること, x が不要であること, 関数が完全未定義であることは, それぞれ次のように表現できる.

ストリクト	$STR \circ f$	$\sqsubseteq$	$f \circ STR$
不要引数	$STR \circ f$	$\sqsubseteq$	$f \circ ABS$
完全未定義	$STR \circ f$	$\sqsubseteq$	$f \circ FAIL$

フラットドメイン上の線形リストの場合には、

$$length(x) = \textbf{if } null(x) \textbf{ then } 0 \textbf{ else } 1 + length(tail(x))$$

$$sumlist(x) = \textbf{if } null(x) \textbf{ then } 0 \textbf{ else } head(x) + sumlist(tail(x))$$

$$until0(x) = \textbf{if } null(x) \textbf{ then } 0 \textbf{ if } head(x) = 0 \textbf{ then } 0 \textbf{ else } head(x) + until0(tail(x))$$

とすると、*length*, *sumlist*, *until0* はそれぞれリストの長さ, 要素の合計, 最初の 0 の要素の手前までの要素の和を計算する関数となり、

$$STR \circ sumlist \sqsubseteq sumlist \circ FIN\ STR,$$

$$STR \circ length \sqsubseteq length \circ FIN\ ID$$

$$STR \circ until0 \sqsubseteq until0 \circ INF\ STR,$$

となり、それぞれトータルストリクト、テールストリクト、ヘッドストリクトな関数であることを表している。

3 Lazy な抽象領域の生成

3.1 遅延不動点計算

抽象解釈における通常の最小不動点計算は、再帰方程式 $\tau : (D \to D) \to (D \to D)$ に対し、完全未定義関数 Ω (つまり、すべての $x \in D$ に対し $\Omega(x) = \perp$) を初期値として、$\tau^n(\Omega)$ が収束するまで計算する [6]。その収束の判定は各 D の元 x に対し $\tau^i(\Omega)(x) = \tau^{i+1}(\Omega)(x)$ を調べることによりなされる。このため、一般には抽象解釈の停止性は D の領域の有限性に依存している。

しかし、通常、必要になるのは最小不動点 $\tau^i(\Omega)$ の特定の D の元 a に対する $\tau^i(\Omega)(a)$ だけであることがしばしばである。計算量の削減や、有限とは限らぬ抽象領域上での抽象解釈のためには、必要な値 $a \in D$ に対する最小不動点の値 $\tau^i(\Omega)(a)$ を lazy に計算することが重要になる。

Lazy な不動点計算において問題となるのは、最小不動点計算が真の収束ではない plateau で収束したと判定してしまわないかどうか、である。たとえば、$D = \{0, 1\}$ $(0 \sqsubseteq 1)$ において $g(0), g(1) = 1$ とし、

$$f(x) = if\ x = 1\ then\ 1\ else\ f(g(x))$$

とするとその不動点計算は

x	$f^0(x)$	$f^1(x)$	$f^2(x)$	$f^3(x)$
0	**0**	**0**	**1**	**1**
1	**0**	**1**	**1**	**1**

となり、真の収束は f^3 で得られるが、$f^i(0)$ のみに注目していると、f^1 の plateau で収束と判定されてしまい、正しい $f(0) = 1$ のかわりに $f(0) = 0$ という誤った答を得てしまう。

lazy に不動点を計算する場合の plateau の問題を避けるためには、以下の判定を用いる。$f^i(a) = f^{i+1}(a)$ の計算において参照される関数と値 $g^j(b)$ $(j \leq i)$ の集合を $Ref(f, i)$、さらに $Ref(f, i)$ の元 $g^j(b)$ から j (g を定義する再帰方程式の unfolding の回数) を除いたものを $ref(f, i)$ とする。このとき、以下の条件が満たされた場合に収束したと判定される。

- $f^i(a) = f^{i+1}(a)$
- $ref(f, i) = ref(f, i + 1)$
- $\forall g^j(b) \in Ref(f, i + 1)$ に対し $g^j(b) = g^{j+1}(b)$

たとえば、上の例では $f^0(\mathbf{0}) = f^1(\mathbf{0})(= \mathbf{0})$ であるが、$f^1(\mathbf{0})$ が参照する $f^0(\mathbf{1})$ において $f^0(\mathbf{1}) \neq f^1(\mathbf{1})$ であるので、収束したとは判定しない。

3.2 遅延抽象領域生成

高階関数型言語のストリクトネス解析においては、抽象領域の自動生成は [1] において提案されているが、それは抽象領域の構成のみを対象としたものであり、ここでは一階の関数型言語に制限した上で、解析の実行時に動的に抽象領域、および抽象解釈に必要な演算を構成する手法の概略について説明する。詳細については次章で例に基づき説明する。

検出したい性質の仕様記述として、再帰領域方程式により代数的に定義された型における定数と関数記号、およびベースとなる領域上の性質と $\sqcup$ により再帰的に構成されるスキーマを用いる。たとえば線形リスト $list(A)$ の場合は、A 上の性質 α に基づき節 2.3 で定義された $FIN\ \alpha$, $INF\ \alpha$ などである。

抽象領域を構成した時、その他に生成する必要のある演算は各基本関数の抽象領域での解釈、$\sqcup$、&、計算過程で生じる性質の包含関係の判定である。

型の定義により、新たに導入される基本関数は、構成子、選択子、識別子であり、たとえば線形リストの場合、それぞれ $nil, cons$、$head, tail$、$null$ である。それぞれの抽象領域での解釈は、構成子、選択子はそのままであり、識別子は冠頭関数記号により適切な Bool 値を対応させる。たとえば nil は NIL、$cons$ は $CONS(ABS, FIN\ STR)$、$head$ は $HEAD(CONS(ABS, FIN\ STR)) = ABS$、$tail$ は $TAIL(CONS(ABS, FIN\ STR)) = FIN\ STR$ などの表現や規則を持ち、$null$ は $NULL(NIL) = true$, $NULL(CONS(x, y)) = false$ により定められる。

$\sqcup$ は、if 文の条件分岐に対応して自然に導入される。

スキーマによって記述された二つの性質の & は、節 2.3 で導入された & や $\sqsubseteq$ に関する規則を書き換え規則とみなし、その変形により、既に生成された性質と等価な再帰的スキーマを満たすかどうか判定する。これについては実例に則して次章で詳述する。

この & の操作が決定可能に計算できるようにするため、現在は代数型の定義および性質記述のスキーマに現われる変

数は一つだけという制限をおく。この場合、変数を特別な定数とみなすことにより、性質記述の再帰方程式の同値性の判定は定項項書換え系の同値性判定（Word Problem、すなわち $M \leftrightarrow^* N$ の判定）に帰着し、同値でないと判定されると新たな抽象領域の元として追加される。（一般には、この手法は同値な性質の同値性を見落とすことはあるが、異なる性質を同値とみなすことはなく、その意味で安全な近似になる。）定項項書換え系の同値性判定問題は決定可能であることが知られている[2]

抽象領域の生成は、検出したい性質の記述を $P_1, \cdots, P_n$, 定数構成子を $C_1, \cdots, C_m$ とした時、初期値を $D = \{P_1, \cdots, P_n, C_1, \cdots, C_m, OTHERS, ABS, FAIL\}$ なる有限抽象領域とおく。ただし、$OTHERS$ は P_i, C_j どれでもでないことを表す $\sqsubseteq$ に関する最大元であり、構成子、選択子により $OTHERS$ は $OTHERS$ に、識別子により $OTHERS$ は $true \sqcup false$ に写像される。

上述のように、新たに型定義にともない導入された構成子（たとえば $CONS$）により、抽象領域は D から自然に生成されるが、無制限に行なうと不動点計算の領域が（lazy に行なっても）無限の領域となり、停止しない抽象解釈となってしまう。抽象解釈の安全性（ある性質を本当は満たしていないのに、満たしていると解析してしまうことがない）を保証しながら停止性も保証するために、各関数の関数定義の 1 回の展開では自由に拡張するが、次の関数定義の展開時に D の元に射影する。この射影の操作は、$P \sqsubseteq Q$ と判定された Q のうち $\sqsubseteq$ に関し最小のものに P を射影することで得られる。$P \sqsubseteq Q$ の判定は、Q の部分項と P の同値性判定に帰着される。

たとえば、$CONS(ABS, FIN\ STR)$ はそれを含む $\sqsubseteq$ に関し最小な $FIN(ABS)$ に射影し、$CONS(STR, NIL)$ は $FIN(STR)$ に射影する。各関数定義内でネストする構成子の数は必ず有限なので、これにより、もとのベース抽象領域（たとえば $\{ABS, STR, ID, FAIL\}$）の有限性により、停止性が保証される。

4 実例

章 3 で概略を説明した lazy な抽象領域の生成を、節 2.3 で導入した $until0$ のヘッドストリクト性 $INF\ STR$ および $sumlist$ のトータルストリクト性 $FIN\ STR$ のプロジェクション解析による検出を例にとり説明する。ここでのリスト型上のスキーマは節 2.3 で導入された $INF\ \alpha$, $FIN\ \alpha$ とする。

$until0$ の関数定義をプロジェクションに関する再帰方程式に変換すると

$$until0\ \alpha = NIL \sqcup CONS(STR, ABS \sqcup until0\ \alpha)$$

となる。ノンフラットドメイン上の抽象領域の初期値は

$$D = \{INF\ STR, NIL, OTHERS, ABS, FAIL\}$$

となり、$until0\ STR$ の計算は以下のようになる。

$$\begin{aligned}
until0^0\ STR &= FAIL \\
until0^1\ STR &= NIL \sqcup CONS(STR, ABS) \\
&\sqsubseteq INF\ STR \\
until0^2\ STR &= NIL \sqcup CONS(STR, INF\ STR \sqcup ABS) \\
&= INF\ STR \quad (収束)
\end{aligned}$$

この結果、$until0$ はヘッドストリクトであることがわかる。また、$sumlist$ についてはプロジェクションに関する再帰方程式は

$$sumlist\ \alpha = NIL \sqcup CONS(STR, sumlist\ \alpha)$$

となる。ノンフラットドメイン上の抽象領域の初期値は

$$D = \{FIN\ STR, NIL, OTHERS, ABS, FAIL\}$$

となり、$sumlist\ STR$ の計算は以下のようになる。

$$\begin{aligned}
sumlist^0\ STR &= FAIL \\
sumlist^1\ STR &= NIL \sqcup CONS(STR, NIL) \\
&\sqsubseteq FIN\ STR \\
sumlist^2\ STR &= NIL \sqcup CONS(STR, FIN\ STR) \\
&= FIN\ STR \quad (収束)
\end{aligned}$$

これらにおいて $NIL \sqcup CONS(STR, ABS) \sqsubseteq INF\ STR$, $NIL \sqcup CONS(STR, NIL) \sqsubseteq FIN\ STR$ は各関数展開時に行なう射影である。また $foo(x) = until0(x) + sumlist(x)$ とすると $foo\ STR = INF\ STR \& FIN\ STR = FIN\ STR$ となるが、これは以下のように計算される。

$$\begin{aligned}
&INF\ STR \& FIN\ STR \\
=\ &(NIL \sqcup CONS(STR, INF\ STR \sqcup ABS)) \\
&\&(NIL \sqcup CONS(STR, FIN\ STR)) \\
=\ &NIL \sqcup CONS(STR, INF\ STR \& FIN\ STR \sqcup FIN\ STR) \\
=\ &\mu X.NIL \sqcup CONS(STR, X \sqcup FIN\ STR)
\end{aligned}$$

ここで、$FIN\ STR$ も再帰的に定義されている（$FIN\ STR = \mu X.NIL \sqcup CONS(STR, FIN\ STR)$）ので、それを除いた方程式 $INF\ STR \& FIN\ STR = \mu X.NIL \sqcup CONS(STR, X)$ を作る。この方程式と $FIN\ STR$ を定義する方程式は一致する（この同値性判定のために定項項書換え系の同値性判定が必要になる）ので、$INF\ STR \& FIN\ STR = \text{FIN STR}$ と判定される。また $goo(x) = until0(x) + length(x)$ とすると $goo\ STR = INF\ STR \& FIN\ ID$ となり、同様に計算すると、

$$\begin{aligned}
&INF\ STR \& FIN\ ID \\
=\ &NIL \sqcup CONS(STR, INF\ STR \& FIN\ ID \sqcup FIN\ ID) \\
=\ &\mu X.NIL \sqcup CONS(STR, X \sqcup FIN\ ID)
\end{aligned}$$

が得られる。再帰方程式で定義される $FIN\ ID$ を消去した方程式 $INF\ STR \& FIN\ ID = \mu X.NIL \sqcup CONS(STR, X)$ は $FIN\ ID$ の性質記述方程式と異なることが同値性判定からわかるので、$INF\ STR \& FIN\ ID \neq FIN\ ID$ が導かれる。さらに $FIN\ ID \not\sqsubseteq INF\ STR$ であるので、$INF\ STR \& FIN\ ID$ は新たなる元として抽象領域に加えられる。

[2] より一般的な場合として、(1) 左線形右定項な場合 [15]、(2) shallow な場合（すなわち書き換え規則の両辺において変数の出現が深さ 1 以下の場合）[4] に同値性判定問題が決定可能であることが知られている。

また $FIN\ ID$ を消去した方程式 $\mu X.NIL \sqcup CONS(STR, X)$ は $FIN\ ID$ の性質記述方程式 $\mu X.NIL \sqcup CONS(ID, X) = \mu X.NIL \sqcup CONS(STR \sqcup ABS, X)$ の部分項と一致することがわかる（ここで定項項書換え系の同値性判定を用いる）ので、$\mu X.NIL \sqcup CONS(STR, X) \sqsubseteq \mu X.NIL \sqcup CONS(ID, X)$ が導かれる。これから $INF\ STR \& FIN\ ID \sqsubseteq FIN\ ID$ が示され、射影により $goo\ STR = FIN\ ID$ となる。

5 まとめ

本稿では、一階の関数型言語を対象とし、解析対象となる性質の仕様をもとに、解析の実行時に動的に抽象領域および抽象解釈に必要な演算を構成する手法の概略について、プロジェクション解析の例に基づき説明した。

しかし現在の手法は、実際には（検証としては）非常に弱い性質しか扱うことができない。その理由は、現在の抽象解釈が対象としているのは、（抽象領域の有限性の制約から）単項述語による性質に限られるためである。

今後の課題は、[11] と同等な検証の抽象解釈での実現可能性、また検証可能な性質を構文的に保証する仕様記述言語である。このためには lazy な抽象領域の生成と不動点計算を用いた場合の無限領域上での抽象解釈の限界（停止性など）を明らかにすることが必要であろう。

また本手法の有効性を検証するためには、実際に自動生成システムとして実装が必要と考えている。

参考文献

[1] P.N. Benton. Strictness properties of lazy algebraic datatypes. In *Proc. 3rd WSA*, pages 206–217, 1993. LNCS 724.

[2] W. Charatonik and L. Pacholski. Set constraints with projections are in *nexptime*. In *Proc. 35th IEEE Sympo. on Foundations of Computer Science*, pages 642–653, 1994.

[3] T.-R. Chuang and B. Goldberg. A syntactic approach to fixed point computation on finite domains. In *Proc. 1992 ACM LFP*, pages 109–118, 1992.

[4] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, cycle-syntacticness, and shallow theories. *Information and Computation*, 111(1):154–191, 1994.

[5] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *Proc. IEEE Int. Conf. on Computer Languages (ICCL '94)*, pages 95–112, 1994.

[6] B.A. Davey and H.A. Priestley. *Introduction to Lattice and Order*. Cambridge University Press, 1990.

[7] K. Davis and P. Wadler. Backwards strictness analysis: Proved and improved. In Davis K. and Hughes J., editors, *in Functional Programming: Proc. 1989 Glasgow Workshop*, pages 12–30. Springer-Verlag, 1990.

[8] C. Hankin and D. Le Métayer. Deriving algorithms from type inference systems: Application to strictness analysis. In *Proc. 21th ACM POPL*, pages 202–212, 1994.

[9] C. Hankin and D. Le Métayer. Lazy type inference for the strictness analysis of lists. In *Proc. ESOP '94*, pages 257–271, 1994. LNCS 788.

[10] J. Hughes and J. Launchbury. Reversing abstract interpretations. *Science of Computer Science*, 22:307–326, 1994.

[11] D. Le Métayer. Proving properties of programs defined over recusive data structures. In *Proc. ACM PEPM '95*, pages 88–99, 1995.

[12] E. Nöcker. Strictness analysis using abstract reduction. In *Proc. Functional Programming Languages and Computer Architecture*, pages 255–265, 1993. ACM Press.

[13] M. Ogawa and S. Ono. Transformation of strictness-related analyses based on abstract interpretation. *IEICE Trans.*, E 74(2):406–416, 1991. Conference Version: *Proc. Int. Conf. Fifth Generation Computer Systems 1988 (FGCS'88)* , pp. 430–438 (1988).

[14] K.M. Olender and L.J. Osterweil. Cecil: A sequencing constraint language for automatic static analysis generation. *IEEE Trans. on Software Engineering*, 16(3):268–280, 1990.

[15] M. Oyamaguchi. On the word problem for right-ground term-rewriting systems. *IEICE Trans.*, E 73(5):718–723, 1990.

[16] R.C. Sekar, P. Mishra, and I.V. Ramakrishnan. Fast strictness analysis based on demand propagation. *ACM TOPLAS*, 17(6):896–937, 1995.

[17] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–192. Elsevier Science Publishers, 1990.

[18] P. Wadler and R.J.M. Hughes. Projections for strictness analysis. In *Proc. Functional Programming Languages and Computer Architecture*, pages 385–407, 1987. LNCS 274.

[19] K. Yi and W.L. Harrison III. Automatic generation and management of interprocedural program analyses. In *Proc. 20th ACM POPL*, pages 246–259, 1993.

[20] 小野諭 小川瑞史. 抽象実行 ーそのフレームワークと実例ー その1〜その3. コンピュータソフトウェア, 13(2,4,6 掲載予定), 1996.

[21] 小川瑞史 小野諭. 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. 情報処理学会論文誌, 32(7):914–923, 1991.