静的かつ動的な式を許すような部分評価器のための束縛時解析

浅井 健一
東京大学大学院 理学系研究科 情報科学専攻
asai@is.s.u-tokyo.ac.jp

1　はじめに

部分評価 (partial evaluation)[10] とは、プログラムと引数の一部を受け取って、その既知の引数の情報から計算できる部分を先に計算してしまうことでプログラムの効率をあげるプログラム変換である。部分評価の技法を使うとインタプリタからコンパイラを生成できる [6]、など応用が広いため、特に関数型言語の世界では多くの研究がなされ、実際にいくつかのシステムが動くに至っている [2]。また、近年ではオブジェクト指向言語の最適化等にも利用されるようになってきている [5, 4]。

部分評価の中でも offline の部分評価と呼ばれる方法では、部分評価をプログラム解析のフェーズ（束縛時解析 BTA と呼ばれる）とプログラム特化のフェーズにわけて行う。解析のフェーズでは、プログラムを部分評価時に実行可能（静的 static）な部分と不可能（動的 dynamic）な部分とにわけ、特化のフェーズでは静的な部分を実行する。この方法は、解析のフェーズであらかじめ実行可能な部分を分離するため特化が効率的に行える一方で、解析の能力が得られる結果に大きな影響を与える、という特徴を持つ。

従来の offline 部分評価器に使われている束縛時解析は、構造データの扱いに難点があった。すなわち、構造データが一度、動的になると、その構成要素が全て動的になってしまっていた。例えば、次の例を見てみよう。

$$(\lambda x.\, cons\;(car\;x)\;x)@(cons\,1\,2)$$

これを部分評価したら $cons\,1\,(cons\,1\,2)$ が得られることを期待するわけだが、従来の束縛時解析では、最終結果に $x (= cons\,1\,2)$ が残っているため、この cons セルは動的となる。すると $car\,x$ も動的となり全く計算が行われないまま、入力した式がそのまま結果として返される[1]。

ここでの問題は、 x が静的にも動的にも使われている、という点である。このような式を、本稿では静的かつ動的 (both static and dynamic) と呼ぶことにする。静的かつ動的な式については、静的な部分は部分評価時に実行しつつ、動的な部分は最終結果に残したい。これを実現する方法として、それぞれの場合に対応すべく 2 バージョン作る、という方法がある。例えば、上の例では $cons\,1\,2$ が静的かつ動的に使われているので、以下のように上のプログラムを変換する。

$$(\lambda x.\, (cons^D (car\;(\mathbf{cdr}\;x))\;(\mathbf{car}\;x)))@(\mathbf{cons}\;(cons^D 1\,2)\;(cons\,1\,2))$$

ここで、肩に D のついているものは動的なもの、それ以外は静的なものである。静的かつ動的な $cons$ に対応して、静的な $cons$ と動的な $cons^D$ の 2 バージョンが作られ、これらふたつが **cons** でまとめられて x に渡されている。そして、静的に使われる時は静的な部分が **cdr** でとられ、動的に残る部分は動的な部分が **car** でとられている[2]。このように静的な部分と動的な部分の両方を用意しておけば、動的な部分を最終結果に残しつつ静的な部分にもアクセスが可能となる。上の例では、もとのプログラムの car を静的に実行でき、希望通り $cons\,1\,(cons\,1\,2)$ を得ることができる。

本稿では、このような部分評価を可能とするため、静的かつ動的な部分を特定する束縛時解析を提案する。この束縛時解析は、型システムとして定式化され、制約を解くことで結果を得る。実際に静的かつ動的に使われている部分のみが静的かつ動的となるため、静的かつ動的な値を作るオーバーヘッドは最小限に抑えられる。

同じ式に複数の束縛時を許す一般的な束縛時解析としては polyvariant な束縛時解析があげられる。しかし、polyvariant な解析はコストが非常に高いことが知られている [1]。ここでは扱える束縛時を静的かつ動的なものまでに限るこ

---

[1] cons セルの x への代入は行われるが、コードの複製を避けるため let 文が挿入され **let** $x = cons\,1\,2$ **in** $cons\,(car\;x)\,x$ が結果となる。

[2] ここでは、見やすくするため **cons**, **car**, **cdr** と書いたが、これらは実際には $cons$, car, cdr を使うことができる。

とで monovariant な解析の効率を損なうことなく、構造データの扱いを向上することを目指している。実際、本稿で示す束縛時解析は、従来のものと同じくほぼ線形でできることがわかっている。現在、この方法に基づいた Scheme 用の処理系を作成中で、ほぼ動くに至っている。

以下、2 節で本稿で扱う言語を説明し、3 節では特化器を示す。4 節で束縛時型を導入した後、5 節に annotation のついた言語に対する型規則を示す。6 節ではこの規則を annotation のついていない言語のものに変換し、制約生成の規則を得る。7 節で、制約解消時に使う型の解釈を示し、8 節で実際に制約解消アルゴリズムを述べる。9 節でそのアルゴリズムの正当性を示すとともに、10 節ではその計算量も述べる。また、簡単な解析例を 11 節に示す。関連研究を 12 節で述べ、13 節でまとめる。

2 言語

本稿で扱う言語は、以下に示すような関数型言語の一部で、高階関数、関数適用、primitive の適用、そして構造データとして $cons$, car, cdr を扱う。ここでは簡単のため再帰を扱っていないが、再帰が入っても本稿の枠組は問題なく拡張できることがわかっている。

$$L \quad = \quad \text{定数} \mid x \mid \lambda x.\, L \mid L @ L \mid \mathit{cons}\, L\, L \mid \mathit{car}\, L \mid \mathit{cdr}\, L \mid L + L$$

言語 L のプログラムを受け取ると、束縛時解析は annotation の付いたプログラムを返す。従来の束縛時解析では、annotation は S（静的）または D（動的）の 2 種類であったが、これに加えて本稿では B（静的かつ動的）も許す。以下に、annotation の付いた言語の定義を示す。

$$\begin{array}{rcl} M & = & \text{定数} \mid x \mid \lambda^S x.\, M \mid \lambda^D x.\, M \mid \lambda^B x.\, M \mid M @^S M \mid M @^D M \mid \\ & & \mathit{cons}^S M\, M \mid \mathit{cons}^D M\, M \mid \mathit{cons}^B M\, M \mid \mathit{car}^S M \mid \mathit{car}^D M \mid \mathit{cdr}^S M \mid \mathit{cdr}^D M \mid \\ & & M +^S M \mid M +^D M \mid M{\downarrow}^B_S \mid M{\downarrow}^B_D \end{array}$$

$\lambda^B x.\, M$ および $\mathit{cons}^B M\, M$ が、静的かつ動的な値を返す命令である。また、$M{\downarrow}^B_S$ と $M{\downarrow}^B_D$ はそれぞれ、静的かつ動的な値を受け取ったら、その静的な部分、動的な部分を返す命令である。このふたつを本稿では型変換命令と呼ぶ。

3 特化器

図 1 に annotation の付いた言語 M に対する特化器を示す。図中の右辺で、肩に $^\diamond$ の付いた変数は新たに導入された fresh な変数、また、肩に D の付いた命令は、それが特化の結果、コードとして残ることを示す。$\xi\kappa. \cdots$ と $\langle\cdots\rangle$ は *shift* と *reset* 命令 [3] である。*shift* 命令は *call/cc* と同じように現在の継続をとって来る命令だが、*call/cc* とは違って、*reset* 命令で囲まれた範囲の継続（部分継続）をとって来る命令である。これらの命令は、コードの複製を避ける $\mathbf{let}^D$ 文を挿入するために用いられている。$\#_1$ と $\#_2$ はペア (A, B) の car 部分、cdr 部分をとって来る命令である。

$$\#_1(A, B) = A, \quad \#_2(A, B) = B$$

特化器 $\mathcal{S}$ は式と環境を受け取ると静的な命令を実行し、動的な命令をコードに残す。静的な命令については普通のインタプリタと同じなので、ここでは動的な命令と静的かつ動的な命令について見ていく。動的な命令は、その命令を表すコードを生成するわけだが、生成されたコードが複製されたり消去されたりするのを避けるため、必ず $\mathbf{let}^D$ 文の挿入が行われる。例えば動的なλ式の場合は、まず、そのλ式の本体部分が部分評価され、その結果が $v^\diamond$ という（新しい）変数に束縛され、$\mathbf{let}^D$ 文に残される。そして、以降の特化（$\langle\kappa v^\diamond\rangle$ の部分）では変数 $v^\diamond$ のみが使われる。$v^\diamond$ の使われ方にかかわりなくλ式が $\mathbf{let}^D$ 文に残っているため、複製も消去もされない仕組みである。ここで κ は *shift* 命令によってとってこられた部分継続である。その後の部分評価の結果を $\mathbf{let}^D$ 文のスコープの中に残すため *shift* 命令で継続を得、それを $\mathbf{let}^D$ 文の本体部分の *reset* 命令の中で適用している。λ式の本体部分 M の特化も *reset* 命令で囲まれているが、これも M の特化の結果のコードが正しく $\lambda^D x^\diamond. \cdots$ のスコープに入るようにするためである。他の動的な命令の特化も同様である。

静的かつ動的な式については、その後の特化に静的な値も動的な値も渡すことになる。例えば、$\lambda^B x.\, M$ の場合は、$\lambda^D x.\, M$ の場合と同じようにλ式を $\mathbf{let}^D$ 文に残す[3]とともに、その後の特化には動的な値（$v^\diamond$）と静的なλ閉包の両方

---

[3] $\lambda^B x.\, M$ と $\mathit{cons}^B M_1\, M_2$ に出てくる $\mathcal{D}(\cdot)$ については後述する。

$$
\begin{array}{rcl}
\mathcal{S} : \mathit{Exp} \rightarrow \mathit{Env} & \rightarrow & \mathit{Residual\ \ Code} \\
\mathcal{S}\,[\![\text{定数}]\!]\,\rho & = & \text{定数} \\
\mathcal{S}\,[\![x]\!]\,\rho & = & \rho(x) \\
\mathcal{S}\,[\![\lambda^S x.\, M]\!]\,\rho & = & \lambda x'.\, \mathcal{S}\,[\![M]\!]\,\rho[x'/x] \\
\mathcal{S}\,[\![\lambda^D x.\, M]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = \lambda^D x^\diamond.\, \langle \mathcal{S}\,[\![M]\!]\,\rho[x^\diamond/x] \rangle \ \mathbf{in}\ \langle \kappa v^\diamond \rangle \\
\mathcal{S}\,[\![\lambda^B x.\, M]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = \lambda^D x^\diamond.\, \langle \mathcal{D}(\mathcal{S}\,[\![M]\!]\,\rho[x^\diamond/x]) \rangle \ \mathbf{in}\ \langle \kappa(v^\diamond, \lambda x'.\, \mathcal{S}\,[\![M]\!]\,\rho[x'/x]) \rangle \\
\mathcal{S}\,[\![M_1 @^S M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho)(\mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![M_1 @^D M_2]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = (\mathcal{S}\,[\![M_1]\!]\,\rho) @^D (\mathcal{S}\,[\![M_2]\!]\,\rho)\ \mathbf{in}\ \langle \kappa v^\diamond \rangle \\
\mathcal{S}\,[\![\mathit{cons}^S M_1\, M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{cons}^D M_1\, M_2]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = \mathit{cons}^D(\mathcal{S}\,[\![M_1]\!]\,\rho)\,(\mathcal{S}\,[\![M_2]\!]\,\rho)\ \mathbf{in}\ \langle \kappa v^\diamond \rangle \\
\mathcal{S}\,[\![\mathit{cons}^B M_1\, M_2]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}\ (m_1, m_2) = (\mathcal{S}\,[\![M_1]\!]\,\rho, \mathcal{S}\,[\![M_2]\!]\,\rho) \\
 & & \quad \mathbf{in}\ \mathbf{let}^D v^\diamond = \mathit{cons}^D \mathcal{D}(m_1)\, \mathcal{D}(m_1)\ \mathbf{in}\ \langle \kappa(v^\diamond, (m_1, m_2)) \rangle \\
\mathcal{S}\,[\![\mathit{car}^S M]\!]\,\rho & = & \#_1(\mathcal{S}\,[\![M]\!]\,\rho) \\
\mathcal{S}\,[\![\mathit{car}^D M]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = \mathit{car}^D(\mathcal{S}\,[\![M]\!]\,\rho)\ \mathbf{in}\ \langle \kappa v^\diamond \rangle \\
\mathcal{S}\,[\![M_1 +^S M_2]\!]\,\rho & = & (\mathcal{S}\,[\![M_1]\!]\,\rho) + (\mathcal{S}\,[\![M_2]\!]\,\rho) \\
\mathcal{S}\,[\![M_1 +^D M_2]\!]\,\rho & = & \xi\kappa.\, \mathbf{let}^D v^\diamond = (\mathcal{S}\,[\![M_1]\!]\,\rho) +^D (\mathcal{S}\,[\![M_2]\!]\,\rho)\ \mathbf{in}\ \langle \kappa v^\diamond \rangle \\
\mathcal{S}\,[\![M{\downarrow}^B_D]\!]\,\rho & = & \#_1(\mathcal{S}\,[\![M]\!]\,\rho) \\
\mathcal{S}\,[\![M{\downarrow}^B_S]\!]\,\rho & = & \#_2(\mathcal{S}\,[\![M]\!]\,\rho)
\end{array}
$$

図 1: 特化器

をペアにして渡す。これらは、その後の特化で必要に応じて型変換命令 $M{\downarrow}^B_D$ や $M{\downarrow}^B_S$ によって動的な部分、静的な部分が取り出されることとなる。ここで「静的かつ動的な値は必ず一度、コードとして $\mathbf{let}^D$ 文の中に残る」ということに注意しよう。このことは「静的かつ動的な関数の引数は動的でなくてはならない」という制限をもたらす。例えば $(\lambda f.\, \mathit{cons}\, f\, (f@2))@(\lambda x.\, x+3)$ という式を考えてみよう。 $\lambda x.\, x+3$ は静的に適用されるとともに最終結果に残るので静的かつ動的である。そこで静的な部分は計算し、動的な部分は残して $\mathit{cons}\,(\lambda x.\, x+3)\, 5$ を結果として得たい。しかし、 $\lambda x.\, x+3$ は一度、動的な値として最終結果に残る必要があるため、 x を静的とすることができない。結果として $\mathit{cons}\,(\lambda x.\, x+3)\,(2+3)$ しか得ることができないのである[4]。

この静的かつ動的なλ式に対する制限は、本稿で扱う束縛時解析が monovariant であることによる。λ式を最終結果に残すためには、そのλ式の本体部分を解析するときに引数に関して何も仮定できないので、引数は動的とせざるを得ない。 monovariant な解析ではプログラム各部にひとつの束縛時しか割り当てられないので、このλ式が別のところで静的に使われていたとしても、その引数は動的とせざるを得ないのである。しかし、このような制限が付いていても、従来、静的かつ動的なλ式は全て動的と扱われてきていたので、従来の部分評価と比べればより部分評価できるようになっている。

一方、関数の本体部分やペアの要素についてはこのような問題はない。例えば、静的かつ動的な関数 $\lambda x.\, \mathit{cons}\, x\, x$ の返り値 $\mathit{cons}\, x\, x$ がどこかで静的に使われたとしよう。そのような場合でも、この $\mathit{cons}\, x\, x$ は静的かつ動的なものとすることで、コードに残しつつ静的にも使用することが可能である。例えば

$$(\lambda^S f.\, \mathit{cons}^D f{\downarrow}^B_D\ (\mathit{car}^S(f{\downarrow}^B_S\ @^S 1)))@^S(\lambda^B x.\, \mathit{cons}^B x\, x)$$

の $\lambda^B x.\, \mathit{cons}^B x\, x$ は静的かつ動的な値を返す静的かつ動的な関数であるが、この式は

$$\mathbf{let}^D u = \lambda^D x.\, \mathit{cons}^D x\, x\ \mathbf{in}\ \mathbf{let}^D v = \mathit{cons}^D 1\, 1\ \mathbf{in}\ \mathit{cons}^D u\, 1$$

のように特化される。

---

[4] x を静的かつ動的とすることも不可能である。静的かつ動的な値は常に静的にも動的にも使用可能な値だが、λ式がコードに残る時の x は動的な情報しか持っていない。

ここで $\lambda^B x.M$ と $cons^B M_1\,M_2$ の規則に出て来る $\mathcal{D}(M)$ は M が動的ならそのまま返し、静的かつ動的なら動的な部分を返す命令である。静的かつ動的なλ式の本体部分は動的でも静的かつ動的でも良いため、コードに残す時に、動的ならそのまま使い、静的かつ動的なら動的な部分を取り出す必要がある。そのため $\lambda^B x.M$ と $cons^B M_1\,M_2$ の規則のコードを残す部分で $\mathcal{D}(\cdot)$ が使われている。なお、ここでは簡単のため特化時に M の束縛時をチェックするように書いているが、 M の束縛時は束縛時解析が終了した時点でわかっているので、それに従って $\lambda^B x.M$ の規則を M が動的な場合と静的かつ動的な場合のふたつにわければ、特化時に束縛時をチェックする必要はなくなる。 $cons^B M_1\,M_2$ も同様で、図 1 の規則は M_1 と M_2 がそれぞれ動的な場合と静的かつ動的な場合の 4 通りを $\mathcal{D}(\cdot)$ を使ってまとめて書いてある。

ここで、静的かつ動的な値を導入してもコードの複製はおきないことを見ておこう。 1 節の例では、静的かつ動的な値を使うとその部分のコードが複製されるように見えるがそのようなことはない。 $\lambda^B x.M$ の規則を見ると M の特化は、コードに残すために 1 回、行われるだけである。その後、そのλ式が静的に使われれば再び特化が起きるが、これは従来の静的なλ式の特化と同じである。また $cons\ ^B M_1\,M_2$ 規則でも、その要素は 1 回だけ特化され、コードに残す際には単に動的な変数を使うだけである。

コードの複製が起きないため、図 1 の特化器は値の id（ポインタ同値性）を保存する。静的かつ動的な値は、静的に使われる時は固有の id を持った静的な値そのものが（コードに残ることなく）使われるため、各値のポインタ同値性は保存される。また、動的に使われる時も固有の変数名が使われるため、ポインタ同値性は保存される。静的な計算と動的な計算はまざらないので全体としてポインタ同値性が保たれていることがわかる。

4　型

本稿で扱う束縛時解析は、型システムとして定式化されており、そこで使われる各式の束縛時の値 π と型 τ は以下のように定義される。

$$\begin{array}{rcl} \pi & = & S \mid D \mid B \\ \tau & = & \pi \mid \tau \to^{\pi} \tau \mid \tau \times^{\pi} \tau \mid \alpha \mid \mu\alpha.\tau \end{array}$$

束縛時値 π は S（静的）、 D（動的）、または B（静的かつ動的）である。これらは同時に定数の型としても用いられる。 $\tau \to^{\pi} \tau$ は束縛時値が π の関数型、 $\tau \times^{\pi} \tau$ は束縛時値が π のペア型である。また α は型変数、 $\mu\alpha.\tau$ は再帰型である。これらは Scheme のような型なし言語で静的な型の付かない項を型付けするのに必要となる[5]。

束縛時型のうち以下の規則で導けるもののみが有効な型である。ここで、 $\Delta \vdash \tau : \pi$ は環境 Δ のもとで型 τ が束縛時値 π を持つことを示す。

$$\Delta \vdash \pi : \pi \qquad \Delta, \alpha : \pi \vdash \alpha : \pi \qquad \frac{\Delta, \alpha : \pi \vdash \tau : \pi}{\Delta \vdash \mu\alpha.\tau : \pi}$$

$$\frac{\Delta \vdash \tau_1 : \pi_1 \quad \Delta \vdash \tau_2 : \pi_2 \quad \boxed{\pi \triangleright' \pi_1 \quad \pi \triangleright \pi_2}}{\Delta \vdash \tau_1 \to^{\pi} \tau_2 : \pi} \qquad \frac{\Delta \vdash \tau_1 : \pi_1 \quad \Delta \vdash \tau_2 : \pi_2 \quad \boxed{\pi \triangleright \pi_1 \quad \pi \triangleright \pi_2}}{\Delta \vdash \tau_1 \times^{\pi} \tau_2 : \pi}$$

標準的な規則に加えて、関数型とペア型には有効な型となるための制限がついている。これらは基本的には従来の束縛時解析に使われる型システムと同じで、例えば「ペアが動的ならその要素も動的」といった類のものである。しかし、それに加えてここでは静的かつ動的な場合も考える必要がある。それを考慮に入れて定義されたのが次の $\pi_1 \triangleright \pi_2$ と $\pi_1 \triangleright' \pi_2$ である。

$$\begin{array}{cccccc} S \triangleright S & S \triangleright D & S \triangleright B & D \triangleright D & B \triangleright D & B \triangleright B \\ S \triangleright' S & S \triangleright' D & S \triangleright' B & D \triangleright' D & B \triangleright' D & \end{array}$$

π_1 が静的なら π_2 は何でも良い。また π_1 が動的なら π_2 は動的でなくてはならない。ここまでは、従来の場合と同じである。 π_1 が静的かつ動的な場合は、 π_2 として静的かつ動的な場合を許すものと許さないものの 2 種類がある。静的かつ動的な関数の引数部分は、動的でなくてはならないので、後者が使われ、それ以外の部分は静的かつ動的な場合も許す前者が使われている。

---

[5] 例えば $\lambda f.\,f@f$ における f の型は $\mu\alpha.\alpha \to \beta$ のようになる。

$$\Gamma \vdash \text{定数} : \boxed{\pi} \qquad \dfrac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda^S x.\, M : \tau_1 \rightarrow^S \tau_2} \qquad \dfrac{\Gamma, x : D \vdash M : D}{\Gamma \vdash \lambda^D x.\, M : D} \qquad \dfrac{\Gamma, x : D \vdash M : \tau \quad \tau \in \{B, D\}}{\Gamma \vdash \lambda^B x.\, M : D \rightarrow^B \tau}$$

$$\Gamma, x : \tau \vdash x : \tau \qquad \dfrac{\Gamma \vdash M_1 : \tau_2 \rightarrow^S \tau_1 \quad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash M_1 @^S M_2 : \tau_1} \qquad \dfrac{\Gamma \vdash M_1 : D \quad \Gamma \vdash M_2 : D}{\Gamma \vdash M_1 @^D M_2 : D}$$

$$\dfrac{\Gamma \vdash M_1 : \tau_1 \quad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash \mathit{cons}^S M_1\, M_2 : \tau_1 \times^S \tau_2} \qquad \dfrac{\Gamma \vdash M_1 : D \quad \Gamma \vdash M_2 : D}{\Gamma \vdash \mathit{cons}^D M_1\, M_2 : D} \qquad \dfrac{\Gamma \vdash M_1 : \tau_1 \quad \Gamma \vdash M_2 : \tau_2 \quad \tau_i \in \{B, D\}}{\Gamma \vdash \mathit{cons}^B M_1\, M_2 : \tau_1 \times^B \tau_2}$$

$$\dfrac{\Gamma \vdash M : \tau_1 \times^S \tau_2}{\Gamma \vdash \mathit{car}^S M : \tau_1} \qquad \dfrac{\Gamma \vdash M : D}{\Gamma \vdash \mathit{car}^D M : D} \qquad \dfrac{\Gamma \vdash M_1 : S \quad \Gamma \vdash M_2 : S}{\Gamma \vdash M_1 +^S M_2 : \boxed{\pi}} \qquad \dfrac{\Gamma \vdash M_1 : D \quad \Gamma \vdash M_2 : D}{\Gamma \vdash M_1 +^D M_2 : D}$$

$$\dfrac{\Gamma \vdash M : D \rightarrow^B \tau \quad \tau \in \{B, D\}}{\Gamma \vdash M{\downarrow}^B_S : D \rightarrow^S \tau} \qquad \dfrac{\Gamma \vdash M : D \rightarrow^B \tau \quad \tau \in \{B, D\}}{\Gamma \vdash M{\downarrow}^B_D : D} \qquad \dfrac{\Gamma \vdash M : B}{\Gamma \vdash M{\downarrow}^B_S : S} \qquad \dfrac{\Gamma \vdash M : B}{\Gamma \vdash M{\downarrow}^B_D : D}$$

$$\dfrac{\Gamma \vdash M : \tau_1 \times^B \tau_2 \quad \tau_i \in \{B, D\}}{\Gamma \vdash M{\downarrow}^B_S : \tau_1 \times^S \tau_2} \qquad \dfrac{\Gamma \vdash M : \tau_1 \times^B \tau_2 \quad \tau_i \in \{B, D\}}{\Gamma \vdash M{\downarrow}^B_D : D} \qquad \dfrac{\Gamma \vdash M : \tau \quad \tau = \tau'}{\Gamma \vdash M : \tau'}$$

図 2: Annotation の付いた言語に対する型規則

有効な型 τ の束縛時値 $\|\tau\|$ は

$$\|\tau\| = \pi \iff \vdash \tau : \pi\text{が導出可能}$$

と定義される。また、型の間には以下のような同値関係を仮定する。

$$D = D \rightarrow^D D = D \times^D D, \quad \mu\alpha.\, \tau = \tau[\mu\alpha.\, \tau / \alpha]$$

5 型規則

図 2 に annotation の付いた言語に対する型規則を示す。静的かつ動的な値の型の有効性にだけ注意しておけば、これらの規則はいずれも標準的なものである。唯一、特殊なのは定数の型と静的な加算の結果の型で、これらはいつも静的とはせず、その使われ方によって何でも良いことにしている。これは静的な定数はいつでも動的な定数に (従来の lift 命令によって) 変換できるからである。また、図 2 の型規則では、定数も静的、動的、静的かつ動的の 3 種類に区別している。これは、定数を他の構造データと同じく統一的に扱うためであるが、静的かつ動的な定数の表現として、静的な定数と動的な定数の両方を持っておく必要はない。静的な定数のみを持っておき、

$$\begin{array}{rcl} \mathcal{S}[\![\text{定数}{\downarrow}^B_D]\!]\rho & = & \mathit{lift}(\text{定数}) \\ \mathcal{S}[\![\text{定数}{\downarrow}^B_S]\!]\rho & = & \text{定数} \end{array}$$

とすることで、従来と同様に定数を扱うことができる。

annotation の付いたプログラム p_{ann} に対し $\vdash p_{ann} : D$ が導けるとき、その annotation は有効である、という。同じプログラムに対する有効な annotation のうちで、以下のふたつの性質を満たすものをより望ましい annotation と呼ぶ。

1. 全ての B を D と読みかえた時に、 D となっている部分がより少ない。

2. 同じ時は、 B となっている部分がより少ない。

第 1 点はより多くの部分を部分評価時に実行できることを示し、第 2 点は部分評価時に静的かつ動的な値を作るオーバーヘッドがより少ないことを示す。第 1 点で B を D と解釈しているのは、静的にしか使われていない値を不必要に静的か

$$\Gamma \vdash \text{定数} : \pi \qquad\qquad \Gamma, x : \tau \vdash x : \boxed{\tau'} \quad \{\boxed{\|\tau\| \geq \|\tau'\|}\}$$

$$\frac{\Gamma, x : \tau_1 \vdash L : \tau_2}{\Gamma \vdash \lambda x.\, L : \tau_1 \to^{\pi} \tau_2} \ \{\pi \triangleright' \|\tau_1\|,\ \pi \triangleright \|\tau_2\|\} \qquad \frac{\Gamma \vdash L_1 : \tau_2 \to^{\pi} \tau_1 \quad \Gamma \vdash L_2 : \tau_2}{\Gamma \vdash L_1 @ L_2 : \boxed{\tau_1'}} \left\{ \begin{array}{ll} \boxed{\|\tau_1\| \geq \|\tau_1'\|}, & \pi \triangleright' \|\tau_2\|, \\ \pi \in \{S, D\}, & \pi \triangleright \|\tau_1\| \end{array} \right\}$$

$$\frac{\Gamma \vdash L_1 : \tau_1 \quad \Gamma \vdash L_2 : \tau_2}{\Gamma \vdash \mathit{cons}\, L_1\, L_2 : \tau_1 \times^{\pi} \tau_2} \ \{\pi \triangleright \|\tau_1\|,\ \pi \triangleright \|\tau_2\|\} \qquad \frac{\Gamma \vdash L : \tau_1 \times^{\pi} \tau_2}{\Gamma \vdash \mathit{car}\, L : \boxed{\tau_1'}} \left\{ \begin{array}{ll} \boxed{\|\tau_1\| \geq \|\tau_1'\|}, & \pi \triangleright \|\tau_1\|, \\ \pi \in \{S, D\}, & \pi \triangleright \|\tau_2\| \end{array} \right\}$$

$$\frac{\Gamma \vdash L_1 : \pi_1 \quad \Gamma \vdash L_2 : \pi_2}{\Gamma \vdash L_1 + L_2 : \pi} \left\{ \begin{array}{l} \pi_1 \triangleright_p \pi_2,\ \pi_2 \triangleright_p \pi_1,\ \pi_1 \triangleright_p \pi,\ \pi_2 \triangleright_p \pi, \\ \pi_1 \in \{S, D\},\ \pi_2 \in \{S, D\} \end{array} \right\} \qquad \frac{\Gamma \vdash L : \tau \quad \tau = \tau'}{\Gamma \vdash L : \tau'}$$

図 3: Annotation の付いていない言語に対する型規則（制約生成規則）

つ動的とし、後でその静的な部分をとってきて使う、といった無駄な場合を排除するためである。また、第 2 点は動的にしか使われていない値を不必要に静的かつ動的とし、後でその動的な部分をとってきてコードに残す、という場合を排除している。

束縛時解析は、 annotation の付いていないプログラム p が与えられた時に、有効な annotation の付いたプログラム p_{ann} のうち、上の意味で最も望ましい annotation を返すもの、ということができる。

6 制約の生成

図 2 の規則は annotation の付いた言語に対する型規則なので、これを使って annotation の付いていないプログラムに annotation を付けることはできない。 annotation の付いていないプログラムに annotation を付けるためには、型規則を annotation の付いていないプログラムに対するものに書き換える必要がある。

この書き換えは（１）同じ命令に対する規則をまとめ、（２）B から S や D への型変換規則を他の規則に埋め込むことで行われる。このうち第 1 点は型の同値関係 $D = D \to^{D} D$ $D = D \times^{D} D$ を使い、第 2 点は型変換規則の入り得る場所を特定することで行われる。書き換えた結果を図 3 に示す。

図中、四角で囲んである部分が型変換が起きる可能性のある場所である。ここに出てくる $\pi \geq \pi'$ という制約は型の変換可能性を示すもので以下のように定義される。

$$B > S, \quad B > D$$
$$\pi \geq \pi' \iff (\pi > \pi') \vee (\pi = \pi')$$

実際に $\pi > \pi'$ となったときには、束縛時解析終了時に π' に対応する項に annotation として $\downarrow^{B}_{\pi'}$ が付く。

型変換がこの 3 ヶ所のみで十分なことは（１）推論規則の結論部分のみで型変換を行えば十分なこと、そして（２）上の 3 ヶ所以外では型変換は無駄であることからわかる。このうち第 2 点は、例えばもしλ式の結論部分で型変換が使われていたら、最初から変換後のλ式を作れば良いことからわかる。

図 3 の規則が図 2 と同じであることを見ていこう。定数の規則は全く同じ、変数の規則も型変換の部分を除けば同じである。λ式の規則は図 2 の 3 つの規則をまとめたものになっている。 $D = D \to {}^{D} D$ であることに注意すれば、 π がそれぞれ S, D, B の場合と図 2 の 3 つの規則が対応し、かつ π に関する制約が無効な型を排除していることがわかる。 cons の規則も $D = D \times^{D} D$ を使えば同様である。

関数適用と car の規則は、静的かつ動的な場合がないので、さらに $\pi \in \{S, D\}$ という制約も加わっている。 primitive の適用のところに出てくる制約は「primitive の引数がひとつでも動的になったら全体も動的となり、その場合は他の引数も動的になる」ということを示している。ここで $\pi_1 \triangleright_p \pi_2$ は「π_1 が動的なら π_2 も動的」ということを示し、次のように定義される。

$$S \triangleright_p S \quad S \triangleright_p D \quad S \triangleright_p B \quad D \triangleright_p D \quad B \triangleright_p S \quad B \triangleright_p D \quad B \triangleright_p B$$

図 3 の規則は annotation の付いていない言語に対する規則となっているが、各式をどういう型にすれば良いかは明らかではない。束縛時解析は、まず図 3 の規則で型に関する制約を生成し、それを解くことで行われる。一度、制約が解消され、束縛時値が求まれば、図 3 の規則が図 2 のどの規則に対応するかはわかるので、annotation を付けることができる。

7　型のペアによる解釈

本稿で行う制約の解消は、まず束縛時値 π を (s,d) という真偽値のペアで表現する。そして π 上の制約をこれらの変数の上の制約に変換し、そこで制約解消を行う。

(s,d) のうち s は値が静的に使われたかどうかを示す真偽値である。具体的には s は（１）静的なλ式が適用された時、（２）静的なペアの *car* または *cdr* がとられた時、そして（３）定数が静的に primitive に適用された時に true になる。逆に $s = \mathsf{false}$ は静的には使われないことを示す。一方 d はその式が最終結果に動的な値として残るかを示す。残る場合には true となり、残らない場合には false となる。

以下の表に s や d がどういう値をとった時に、その束縛時値が何になるかをまとめた。s のみが真の場合は、その値は使われるが最終結果には残らないので静的である。また、逆に d のみが真の場合は、最終結果に残るだけで使われないので動的である。$s = d = \mathsf{true}$ の時は、その値は静的に使われるとともに最終結果にも残ることを示している。この場合は静的かつ動的となる。一方、$s = d = \mathsf{false}$ の場合は、その値が静的に使われることもなければ最終結果にも残らないことを示す[6]。この場合は、その値をどう扱っても結果には影響しないのだが、ここでは便宜的に $\perp$ とした。

	$d = \mathsf{false}$	$d = \mathsf{true}$
$s = \mathsf{false}$	$\perp$	D
$s = \mathsf{true}$	S	B

このように束縛時値を解釈すると、より望ましい annotation とは

1. d が true（つまり D または B）となっているものがより少ない。
2. 同じ時は、s が true となっているものがより少ない。

と解釈できる。

(s,d) を使って、図 3 に出てくる π 上の制約を以下のように (s,d) 上の制約に変換する。（以下では $\pi = (s,d)$, $\pi_i = (s_i,d_i)$ とする。）

$$\begin{array}{rcl}
\pi_1 \triangleright \pi_2 & : & (d_1 = \mathsf{true}) \Rightarrow (d_2 = \mathsf{true}) \wedge (s_1 = \mathsf{false}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi_1 \triangleright' \pi_2 & : & (d_1 = \mathsf{true}) \Rightarrow (d_2 = \mathsf{true}) \wedge (d_1 = \mathsf{true}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi_1 \triangleright_p \pi_2 & : & (s_1 = \mathsf{false}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi_1 \geq \pi_2 & : & (d_2 = \mathsf{true}) \Rightarrow (d_1 = \mathsf{true}) \wedge (s_1 = \mathsf{false}) \Rightarrow (s_2 = \mathsf{false}) \\
\pi \in \{S, D\} & : & (s = \mathsf{false}) \Rightarrow (d = \mathsf{true}) \wedge (d = \mathsf{true}) \Rightarrow (s = \mathsf{false})
\end{array}$$

いずれの制約も（型が $\perp$ となる場合を除けば）もとの制約と同じ制約を表している。このことは (s,d) の真偽値の全ての場合を確かめれば確認できる。直観的には以下のようになる。$\pi_1 \triangleright \pi_2$ は「π_1 が最終結果に残るなら π_2 も残る」、$\pi_1 \triangleright' \pi_2$ は「π_1 が最終結果に残るなら π_2 は動的でなくてはいけない」、$\pi_1 \triangleright_p \pi_2$ は「π_1 が静的に使われないなら π_2 も使われない」、$\pi_1 \geq \pi_2$ は（s についての制約は対偶をとって）「π_2 が静的に使われるのなら π_1 も使われ、π_2 が最終結果に残るなら π_1 も残る」となる。

8　制約の解消

プログラムが与えられたら前節までの方法に従って制約を生成する。そして、そのプログラム自体の型を $\pi = (s,d)$ とすれば $d = \mathsf{true}$ という制約を加えた上で制約解消を行う。この $d = \mathsf{true}$ という制約は、このプログラムを部分評価した結果自体が最終結果に残らなくてはいけない、というもので、初期条件のようなものである。

制約の解消は、より望ましい annotation の 2 条件にそって以下の 2 段階で行う。

---

[6] 例えば $car\ (cons\ 1\ 2)$ の 2 など。

第 1 段階　まず（初期条件に使われる d を除く）全ての d に false、全ての s に true を代入する。そして、制約 $(A) \Rightarrow (B)$ を「A が成り立っていたら B が成り立つように値を変更する」と解釈して適用し、これを変化がなくなるまで続ける。

第 2 段階　d の値は第 1 段階で確定したものを使用し、s の値は全て false として制約の適用を変化がなくなるまで行う。この際、制約は全て対偶をとって解釈する。つまり $(A) \Rightarrow (B)$ を「B が成り立っていなかったら A が成り立たないように値を変更する」と解釈する。

第 1 段階では $s =$ true, $d =$ false すなわち全ての値をまず S と仮定し、その中で $d =$ true すなわち最終結果に残すため B としなくてはならないものを特定する。第 1 段階で d だけではなく s も考慮に入れているのは s の値が d の値に影響を及ぼすことがあるからである。それは、具体的には（関数適用など）$\pi \in \{S, D\}$ の制約が現れるところである。関数適用は S か D のどちらかでなくてはならないが、S とはなり得ない場合にのみ D としたい。そこで、まず S または B（つまり $s =$ true）という仮定からはじめ、S または B とはなり得ない（$s =$ false）ことがわかったときのみ D（または $\perp$）とするのである。

ここで「関数適用等は可能なら S としたい」という要求と「D としてしか使われていない値を不必要に B にはしたくない」という要求は相容れないことに注意しよう。前者は s をなるべく true にし、後者は s をなるべく false にする方向である。このため、本稿の制約解消アルゴリズムは 2 段階に分かれている。ふたつの要求のうちより重要な前者が第 1 段階で満たされ、後者の要求は第 2 段階で満たされる。

第 2 段階では、第 1 段階で B となったもののうち、実際に静的かつ動的に使われているものを特定する。そのためにまず、第 1 段階で B（または D）と判断したものを全て D として、B としては使われていないと仮定する。そして、その中で S の計算を行うために必要なものを B に戻すことで、実際に静的かつ動的に使われているものを特定している。

9　アルゴリズムの正当性

ここでは前節のアルゴリズムが実際、期待通り動くことを見ていく。以下では真偽値の間の大小関係を true $>$ false とする。第 1 段階は、$s =$ true, $d =$ false から始めて、必要に応じて s を false に、d を true にする。次の補題は制約の適用を任意の順番で行って良いことを示している。

補題 1　第 1 段階の制約の適用は $(s_1, d_1) \geq (s_2, d_2) \iff (s_1 \leq s_2) \wedge (d_1 \geq d_2)$ という束のもとで型を減少させない。また、制約は一度、満たされたら以後、第 1 段階の間はずっと満たされたままである。

証明：　制約 $(A) \Rightarrow (B)$ の B の部分には $s =$ false あるいは $d =$ true という形しか現れていない。もとの s や d の値が何であっても、これらの値の代入では型が小さくなることはない。また、一度、この制約が満たされたら、以後 $s =$ true や $d =$ false が代入されることはないので、制約はずっと満たされたままである。 □

極端な場合として全ての s が false、d が true の場合を考えると、自明に全ての制約を満たしているので、解は存在する。第 1 段階のアルゴリズムは、最小の解から必要なもののみについて型を持ち上げることで制約を満たしているので、いずれ最小の解に到達する。従って、結果は期待通り制約を満たすもののうちで d が true となっているものが最も少ないものとなっていることがわかる。

次に第 2 段階である。今度は $s =$ true、d は第 1 段階で得られたものに設定して制約を適用する。次の補題は、第 2 段階では d はもはや変化しないことを示す。

補題 2　第 2 段階では d の値が更新されることはない。

証明：　第 1 段階終了時には、全ての制約が満たされているので $(d =$ true$) \Rightarrow (d' =$ true$)$ の形の制約は全て満たされている。従って、これらが d の値を書き換えることはない。また d に言及していない制約は d の値を書き換えることはない。書き換える可能性のある規則は $\pi_1 \triangleright' \pi_2$ と $\pi \in \{S, D\}$ の規則に出てくる $(d =$ true$) \Rightarrow (s =$ false$)$ の対偶の $(s =$ true$) \Rightarrow (d =$ false$)$ の形の制約を適用しようとした時に、$d =$ true だった場合のみである。

第 2 段階の開始時には全ての s は false なので s が true になる可能性のあるのは $\pi \in \{S, D\}$ の規則に出てくる $(s =$ false$) \Rightarrow (d =$ true$)$ の対偶の $(d =$ false$) \Rightarrow (s =$ true$)$ による場合と、その後 $(s =$ false$) \Rightarrow (s' =$ false$)$ の対偶の $(s' =$ true$) \Rightarrow (s =$ true$)$ の制約で $s =$ true が伝搬してきた場合のみである。今、第 1 段階の結果 $d_0 =$ false で、それ

が $(d_0 = \mathsf{false}) \Rightarrow (s_0 = \mathsf{true})$, $(s_{i-1} = \mathsf{true}) \Rightarrow (s_i = \mathsf{true})$ $(1 \leq i \leq n)$, $(s_n = \mathsf{true}) \Rightarrow (d_n = \mathsf{false})$ というように、伝搬して d_n に false を代入しようとしたとしよう。このとき第 1 段階終了時の d_n の値が true であることはない。なぜならもし true だったら、上の経路を逆にたどって第 1 段階で $d_0 = \mathsf{true}$ となっていたはずだからである。

以上により d の値は第 2 段階では更新されないことが証明された。 □

これにより、第 2 段階では s についてのみ考えれば良い。これは「第 2 段階では無駄な B を D にする」という直観にも合致している[7]。

次の補題は、第 2 段階における制約の適用も任意の順番で行って良いことを示している。ここで、束の定義で s の大小関係が第 1 段階とは逆になっていることに注意しよう。

補題 **3** 第 2 段階の制約の適用は $(s_1, d_1) \geq (s_2, d_2) \iff (s_1 \geq s_2) \wedge (d_1 = d_2)$ という束のもとで型を減少させない。また、制約は一度、満たされたら以後、ずっと満たされたままである。

証明： d の値は変わらないので s のことだけを考えれば良い。制約の対偶の結論部分で s が出てくるのは全て $s = \mathsf{true}$ という形をしている。もとの s の値が何であっても、この代入によって型が小さくなることはない。また、一度、この制約が満たされたら、以後 $s = \mathsf{false}$ が代入されることはないので、制約はずっと満たされたままである。 □

極端な場合として、第 1 段階の結果を持ってくれば制約は全て満たされているので、解は存在する。 s について最小の解から始めて、必要なもののみを持ち上げることで制約を満たしているので、いずれ最小の解に到達する。 d の値は第 1 段階で得られるものと同じなので、第 2 段階の結果は期待通り、制約を満たすもののうちで d が true となっているものが最も少なく、その中で s が true となっているものが最も少ないものとなっていることがわかる。

10 計算量

ここでは、アルゴリズムがプログラムの長さに対してほぼ線形時間であることを見ていく。まず、生成される制約の数は各命令に対して定数なので、プログラムの長さに対して線形である。しかし、制約を生成する過程で型の単一化が起きるため、その時に union/find アルゴリズム [12] が必要となる。 union/find アルゴリズムの計算量は、プログラムの大きさを N とし、アッカーマン関数の逆関数を α とすると $O(N\alpha(N,N))$ であるが、実際的なプログラムに対しては $\alpha(N,N)$ は 4 以下であることがわかっている [12] ので、ほぼ線形と言うことができる。

第 1 段階は、初期条件の $d = \mathsf{true}$ から始めて、以後 $d = \mathsf{true}$ あるいは $s = \mathsf{false}$ を順に伝搬していくことで線形時間で終了する。これは、各制約が一度、満たされたらその後はずっと満たされ続けるため、高々制約の数しか伝搬する必要がないためである。第 2 段階も同様で、 $d = \mathsf{false}$ であるもの全てについて s の値を true にするか調べ、それを伝搬していくことで、線形時間で終了する。

11 例

1 節の例を本稿で述べた束縛時解析にかけた結果を以下に示す。この結果を見ると、正しく $x = \mathit{cons}\,1\,2$ が静的かつ動的な値として認識され、四角で囲んだ部分でそれぞれその静的な部分、動的な部分が取り出されていることがわかる。

$$\dfrac{\dfrac{\dfrac{\dfrac{\boxed{x : D \times^B D \vdash x : D \times^S D}}{x : D \times^B D \vdash \mathit{car}\, x : D} \qquad \dfrac{\boxed{x : D \times^B D \vdash x : D \times^D D} \quad D \times^D D = D}{x : D \times^B D \vdash x : D}}{x : D \times^B D \vdash \mathit{cons}\,(\mathit{car}\, x)\, x : D}}{\vdash \lambda x.\, \mathit{cons}\,(\mathit{car}\,x)\,x : D \times^B D \to^S D} \qquad \dfrac{\vdash 1 : D \quad \vdash 2 : D}{\vdash \mathit{cons}\,1\,2 : D \times^B D}}{\vdash (\lambda x.\, \mathit{cons}\,(\mathit{car}\,x)\,x)@(\mathit{cons}\,1\,2) : D}$$

これに対応してもとのプログラムに annotation を付けると

$$(\lambda^S x.\, \mathit{cons}^D (\mathit{car}^S x{\downarrow}^B_S)\, x{\downarrow}^B_D)@^S(\mathit{cons}^B 1\,2)$$

となる。型変換の行われている部分（四角で囲まれた部分）に対応して型変換命令が挿入されているのがわかる。

---

[7] 加えて、無駄な S を $\perp$ にもしている。

12 関連研究

本研究は、Hornof[9] らが C で行った束縛時解析と密接な関係がある。彼らは、抽象解釈に基づいて構造データの使用と定義を前向き解析と後向き解析を使って求めている。これは、ちょうど本稿で s と d を使って 2 段階で制約を解消しているのに相当している。本研究では、彼らの仕事を型システムとして定式化するとともに、高階関数についても制限付きながら同様のことが行えることを示した。

本稿で示した型システムと制約解消系は、Henglein[8] の作成した従来の束縛時解析を静的かつ動的な束縛時を許すように拡張したものと見ることができる。静的かつ動的な値が入ったことで制約解消は 2 段階になったが、制約解消の計算量は従来のものと同じくほぼ線形で行うことができる。

Sperber[11] は、静的、動的、そして未知、という 3 つの束縛時を扱う部分評価器を作成している。これは、静的、動的以外の値を扱う、という点では本研究と類似しているが、内容的には独立である。Sperber の部分評価器は、条件分岐のそれぞれの束縛時が異なる時に、その条件文の束縛時を動的とはせずに未知としておき、online にその束縛時を判断する、というものである。このような場合には、我々の部分評価器では動的と判断される。一方、静的かつ動的な値というのは、静的か動的かわからないのではなく、必ず、静的にも動的にも使われるような値である。このような値は、Sperber の方法では動的となる。

13 おわりに

本稿では、関数型言語の部分評価器で、構造データの扱いを向上すべく、静的かつ動的な値を許すような束縛時解析を提案した。この束縛時解析は、型システムとして定式化され、制約を解くことで結果を得る。さらに、型システムの正当性を示すとともに、制約の解消が従来の効率的な束縛時解析と同じくほぼ線形時間でできることを示した。これに基づいて処理系を作成中であり、現在、ほぼ動くに至っている。今後の課題としては、実際に大きなプログラムを使った部分評価の実験と評価、枠組の多段の部分評価 [7] への拡張などがあげられる。

謝辞　日頃より本研究について多くのコメントを頂いている増原英彦氏に感謝いたします。

参考文献

[1] Ashley, J. M., and C. Consel “Fixpoint Computation for Polyvariant Static Analyses of Higher-Order Applicative Programs,” *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 5, pp. 1431–1448 (September 1994).

[2] Bondorf, A. *Similix 5.0 Manual*, DIKU, University of Copenhagen (May 1993).

[3] Danvy, O., and A. Filinski “Abstracting Control,” *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160 (June 1990).

[4] Dean, J., C. Chambers, and D. Grove “Identifying Profitable Specialization in Object-Oriented Languages,” *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM ’94)*, pp. 85–96 (June 1994).

[5] 藤波 順久「オブジェクト指向言語を利用した自動的かつ効率的な実行時コード生成」コンピュータソフトウェア Vol. 15, No. 5, pp. 25-37 (September 1998).

[6] Futamura, Y. “Partial evaluation of computation process – an approach to a compiler-compiler,” *Systems, Computers, Controls*, Vol. 2, No. 5, pp. 45–50, (1971).

[7] Glück, R., and J. Jørgensen “An Automatic Program Generator for Multi-Level Specialization,” *Lisp and Symbolic Computation*, Vol. 10, No. 2, pp. 113–158, Kluwer Academic Publishers (July 1997).

[8] Henglein, F. “Efficient Type Inference for Higher-Order Binding-Time Analysis,” In J. Hughes, editor, *Functional Programming Languages and Computer Architecture (LNCS 523)*, pp. 448–472 (August 1991).

[9] Hornof, L., C. Consel, and J. Noyé “Effective Specialization of Realistic Programs via Use Sensitivity,” In Van Hentenryck P., editor, *Static Analysis (LNCS 1302)*, pp. 63-73 (1997).

[10] Jones, N. D., C. K. Gomard, and P. Sestoft *Partial Evaluation and Automatic Program Generation*, New York: Prentice-Hall (1993).

[11] Sperber, M. “Self-Applicable Online Partial Evaluation,” In O. Danvy, R. Glück, and P. Thiemann editors, *Partial Evaluation (LNCS 1110)*, pp. 465–480 (February 1996).

[12] Tarjan, R. *Data Structures and Network Flow Algorithms*, Volume CMBS 44 of *Regional Conference Series in Applied Mathematics*, SIAM (1983).