継続計算のための仮想機械の導出

木谷有沙　(指導教官: 浅井健一)

1　はじめに

プログラム変換によって単純λ計算のインタープリタから抽象機械を機械的に導出する手法 [2] と、抽象機械をコンパイラと仮想機械に分割する手法 [1] は、ともに先行研究によって示されている。本研究では、まず単純λ計算のインタープリタを継続渡し形式に変換して shift/reset をサポートするインタープリタを導出し、それに対してプログラム変換を行うことで、継続計算のための抽象機械を導出する。そして、その抽象機械を分割することによって継続計算のためのコンパイラと仮想機械を得られることを示す。

2　インタープリタの CPS 変換

継続（計算が終わった後にする処理, continuation）を引数として渡すことで、全ての関数を末尾呼び出しにする形式のことを、継続渡し形式（CPS : Continuation Passing Style）と呼ぶ。単純λ計算のインタープリタを CPS にするには、評価器が評価する項（term）や環境（environment）に加えて、継続を受け取るようにする。ここでの継続は、「値（value）を受け取ったら、次にすべき評価をする」という関数となる。そして、評価器のうち値を返していた節において、値をそのまま返すのではなく継続に値を渡すようにする。

3　shift / reset

shift/reset は、プログラム中の限定された区間の継続を操作するオペレータである。今回の実装では、プログラム中で shift(...) と書く（... の部分には関数が入る）とユーザプログラムの中から現在の継続を切り取ってきて、括弧内の関数に継続を引数として渡し、その後関数のボディ部分を実行する。プログラム中で reset(...) と書くと、括弧内の shift 命令で取って来られる継続を、括弧内の項で行われる評価のみに限定する。

4　SECD マシンと SECC マシン

SECD マシンとはλ計算式を評価する仮想機械で、「値を貯めるスタック (stack)」「環境」「コード (code)」「ダンプ (dump)」の四つ組の状態遷移操作の繰り返しとして定義されている。ダンプとは他の三つ組を一時的に保存するための領域で、関数呼び出しの際にダンプにその時のスタック、環境、コードが保存され、関数のボディの処理後にダンプから戻される。また、J Operator という継続計算のオペレータではダンプの中身をそのまま継続として取ってくる。これら以外でダンプが使われることはない。よって、関数呼び出し後にスタック、環境、コードを適切な状態に復元でき、かつ J Operator を実装しないのであれば、「スタック」「環境」「コード」の三つ組の状態遷移により仮想機械を構成することができる。よって本研究では、ほぼ SECD マシンに近い挙動をするがダンプを持たない仮想機械を導出する。その上で、shift/reset での継続の切り取りをコードのコピーとして明示するために、コードのリストを評価器に受け渡すようにする。関数呼び出しの際にスタックとコードは退避されず、環境はスタックに保存される。コードリストへのコードの退避は reset 命令が行われたときにのみ行われる。この仮想機械は「スタック」「環境」「コード」「コードリスト」の四つ組の状態遷移操作の繰り返しとして定義されるので、SECC マシンと呼ぶことにする。

5　抽象機械の導出

以下に示す手順で抽象機械を導出する。

(1) CPS 変換

評価器を CPS 変換し、末尾再帰にする。この変換により、項の評価順序が実装言語の評価戦略に依存しなくなる。ただし本研究では shift/reset の実装のため既に一度 CPS 変換されたインタープリタを扱うため、ここでの CPS 変換は行わない。

(2) 値と環境をスタックで受け渡しするよう変更

この変換により、それまではシステム上で各関数のスコープに保存されていた値や環境が評価器内で明示的に受け渡しされるようになる。ここで、関数呼び出し後の継続において環境が自由変数になるのを防ぐため、関数呼び出しの際に環境をコピーし、スタックに保存している。

(3) 非関数化

CPS 変換により評価器内でクロージャを生成するようになっているのを、クロージャを使わずに同等の処理を行うプログラムに変換する。この変換により、評価器が高階関数でなくなる。継続が関数ではなく項のリストになるため、評価器の引数として新たに「項のリストのリスト」を受け渡すように変更することによって、reset 命令での継続の限定範囲を判別するようにした。また関数呼び出し後の環境の復帰のために、環境をスタックから出し入れするための項を新たに定義し、評価中に必要な箇所でコードへ追加するようにした。

(4) 関数をまとめる

それぞれ一変数について再帰的な処理を行っていた四つの関数を、ひとつの関数にまとめる。

得られる抽象機械の状態は四つ組 $\langle s, e, c, cl \rangle$ であり、

c	$\Rightarrow$	$\langle [],\ [],\ \mathrm{Term}(c) :: [],\ [] \rangle$
$\langle s,\ e,\ \mathrm{Term}(x) :: c,\ cl \rangle$	$\Rightarrow$	$\langle e(x) :: s,\ e,\ c,\ cl \rangle$
$\langle s,\ e,\ \mathrm{Term}(\lambda x.t) :: c,\ cl \rangle$	$\Rightarrow$	$\langle \mathrm{VFun}(x, t, e) :: s,\ e,\ c,\ cl \rangle$
$\langle s,\ e,\ \mathrm{Term}(t_0\ t_1) :: c,\ cl \rangle$	$\Rightarrow$	$\langle s,\ e,\ \mathrm{Term}(t_1) :: \mathrm{Term}(t_0) :: \mathrm{Apply} :: c,\ cl \rangle$
$\langle s,\ e,\ \mathrm{Term}(shift(t)) :: c,\ cl \rangle$	$\Rightarrow$	$\langle \mathrm{VCont}(s, e, c) :: [],\ e,\ \mathrm{Term}(t) :: \mathrm{Apply} :: [],\ cl \rangle$
$\langle s,\ e,\ \mathrm{Term}(reset(t)) :: c,\ cl \rangle$	$\Rightarrow$	$\langle [],\ e,\ \mathrm{Term}(t) :: [],\ (\mathrm{CSetEnv}(s, e) :: c) :: cl \rangle$
$\langle \mathrm{VFun}(x, t, e') :: v :: s,\ e,\ \mathrm{Apply} :: c,\ cl \rangle$	$\Rightarrow$	$\langle \mathrm{VEnv(e)} :: s,\ (x, v) :: e',\ \mathrm{Term}(t) :: \mathrm{CPopEnv} :: c,\ cl \rangle$
$\langle \mathrm{VCont}(s', e', c') :: v :: s,\ e,\ \mathrm{Apply} :: c,\ cl \rangle$	$\Rightarrow$	$\langle v :: s',\ e',\ c',\ (\mathrm{CSetEnv}(s, e) :: c) :: cl \rangle$
$\langle s,\ e,\ \mathrm{CSetEnv}(s', e') :: c,\ cl \rangle$	$\Rightarrow$	$\langle s\ @\ s',\ e',\ c,\ cl \rangle$
$\langle v :: \mathrm{VEnv}(e') :: s,\ e,\ \mathrm{CPopEnv} :: c,\ cl \rangle$	$\Rightarrow$	$\langle v :: s,\ e',\ c,\ cl \rangle$
$\langle s,\ e,\ [],\ c :: cl \rangle$	$\Rightarrow$	$\langle s,\ e,\ c,\ cl \rangle$
$\langle v :: s,\ e,\ [],\ [] \rangle$	$\Rightarrow$	v

図 1: 抽象機械の状態遷移規則

c	$\Rightarrow$	$\langle [],\ [],\ c,\ [] \rangle$
$\langle s,\ e, \mathrm{access}(x); c,\ cl \rangle$	$\Rightarrow$	$\langle e(x) :: s,\ e,\ c,\ cl \rangle$
$\langle s,\ e, \mathrm{close}(x, c'); c,\ cl \rangle$	$\Rightarrow$	$\langle \mathrm{VFun}(x, c', e) :: s,\ e,\ c,\ cl \rangle$
$\langle s,\ e, \mathrm{shift}(c'); c,\ cl \rangle$	$\Rightarrow$	$\langle \mathrm{VCont}(s, e, c) :: [],\ e,\ c',\ cl \rangle$
$\langle s,\ e, \mathrm{reset}(c'); c,\ cl \rangle$	$\Rightarrow$	$\langle [],\ e,\ c',\ (\mathrm{setEnv}(s, e) :: c) :: cl \rangle$
$\langle \mathrm{VFun}(x, c', e') :: v :: s,\ e, \mathrm{apply}; c,\ cl \rangle$	$\Rightarrow$	$\langle s,\ (x, v) :: e',\ c'\ @\ c,\ cl \rangle$
$\langle \mathrm{VCont}(s', e', c') :: v :: s,\ e, \mathrm{apply}; c,\ cl \rangle$	$\Rightarrow$	$\langle v :: s',\ e',\ c',\ (\mathrm{setEnv}(s, e) :: c) :: cl \rangle$
$\langle s,\ e, \mathrm{setEnv}(s', e'); c,\ cl \rangle$	$\Rightarrow$	$\langle s\ @\ s',\ e',\ c,\ cl \rangle$
$\langle v :: \mathrm{VEnv}(e') :: s,\ e, \mathrm{popEnv}\ ; c,\ cl \rangle$	$\Rightarrow$	$\langle v :: s,\ e',\ c,\ cl \rangle$
$\langle v_1 :: v_0 :: s,\ e, \mathrm{pushEnv}; c,\ cl \rangle$	$\Rightarrow$	$\langle v_1 :: v_0 :: \mathrm{VEnv}(e) :: s,\ e,\ c,\ cl \rangle$
$\langle s,\ e,\ nil,\ c :: cl \rangle$	$\Rightarrow$	$\langle s,\ e,\ c,\ cl \rangle$
$\langle v :: s,\ e,\ nil,\ [] \rangle$	$\Rightarrow$	v

図 2: 仮想機械の状態遷移規則

その状態遷移規則は図 1 のように定義される。

6 仮想機械の導出

前節で得た抽象機械を、Ager [1] らの手法を用いてコンパイラと仮想機械へと分割する。前節で得た評価器は「項を引数とし、(再帰的に処理を行い、) コードを返す」関数と、「スタック、環境、コード、コード列を引数とし、値を返す」関数に分割でき、前者をコンパイラ、後者を仮想機械と見なすことができる。実際には以下の手順でコンパイラ及び仮想機械の導出を行った。

(1) 評価器を、上述の考えに基づきカリー化する。

(2) 中間言語 (仮想機械にとっての機械語) を定義し、(1) をもとに、項を受け取ったらそれをコードに変換するコンパイラを定義する。また、スタック、環境、コードを受け取ったら値を返す仮想機械を構成する。このとき、環境を出し入れするための命令はコンパイラ側で追加するように変更し、仮想機械でコードを処理する際の、コードへの干渉を減らした。

こうして得られる仮想機械の状態は四つ組 $\langle s, e, c, cl \rangle$ であり、その状態遷移規則は図 2 のように定義される。

7 まとめと今後の課題

本研究では shift/reset をサポートしたインタープリタについて、機械的なプログラム変換によってコンパイラ及び仮想機械を得られることを示した。今後は本研究で得られた仮想機械の命令列からアセンブラへ変換する手法について検討し、最終的には機械語のレベルで限定継続の処理がどのように扱われるのかを明らかにすることを目指す。そしてそれに伴い、継続計算のための処理系の系統的な導出法を示したい。

参考文献

[1] M. S. Ager, D. Biernacki, O. Danvy, and J. Midtgaard: "From Interpreter to Compiler and Virtual Machine : A Functional Derivation," Technical Report RS-03-14, BRICS, Aarhus, Denmark (March 2003).

[2] O. Danvy and K. Millikin: "A Rational Deconstruction of Landin's J Operator," Technical Report RS-06-17, BRICS, Aarhus, Denmark (December 2006).