無限オブジェクトの実行順序の分類

2013年5月28日 Revision 1.00

古田秀和 (FURUTA Hidekazu)

背景

オブジェクト指向の特長として、「永久に存在し続けるオブジェクト」を表すことが容易で、このためGUIを記述することなどが容易にできるということがあります。オブジェクト指向言語(C++, C#, Java, Smalltalkなど)では、ポインタの受け渡しのような仕組みによってこのようなオブジェクトが実現されると考えられます。この文章ではこのようなオブジェクトを「無限オブジェクト」を呼ぶことにします。

オブジェクト指向のライブラリは個別に対応したりすることは容易ですが、個別のものを再利用することは難しく、そのためには何か系統的な変換の方法が必要と考えられます。オブジェクト指向的な機能と、関数型言語的な機能を持つ言語があるのはこの問題の解決のためと考えられます。プログラミング言語F#ではオブジェクト指向のライブラリを使うことができるのですが、そのためにはオブジェクト指向的な機能を使う必要があります。

関数型言語的にオブジェクト指向ライブラリを使うためには、そのための入り口がライブラリに必要なのではないかと思われます。関数型言語的にも評価の順序が決まっているもの(引数の評価の後に関数の評価が行われる)、決まっていないものがあり、それによって「無限の構造を持つもの」(こちらも「無限オブジェクト」と呼ぶことにします)の扱い方も異なります。これも入り口を分ける必要があると考えられます。

この文章では論理プログラミング言語を使って「無限オブジェクト」の分類を試みます。

論理プログラミング言語の定義

まず、有限の構造だけを扱うことを考えます。プログラミング言語PrologやGHC(Guarded Horn Clauses)を参考にして
並行論理プログラミング言語を定義します。この内容はPrologについてご存じであればわかりやすいと思います。まず論理プログラミングの考え方について述べ、後でPrologの構文と実行順序についても述べます。

CLP^{0}の項の定義

まず有限の構造を扱うCLP^{0}を定義します。(CLPはConcurrent Logic Programmingの意味)

V を変数の集合(可算)、C をコンストラクタ(定数も含む)の集合(有限)、F を関数の集合とします。各コンストラクタ c および関数 f には、アリティ(項の個数)を表す自然数(0以上)が対応しています。項は変数、「コンストラクタの項」、「関数の項」のどれかで、「コンストラクタの項」はコンストラクタ(アリティを n とする)と n 個の項の組、「関数の項」は関数(アリティを n とする)と n 個の項の組、と再帰的に定義されます。(項の集合 T は $T = V + C \times T^* + F \times T^*$ とすることもできます。コンストラクタ、関数に対して一つのアリティが決まります。ここで集合 X に対して $X^n = X \times X \times \ldots \times X$ (n 個)、$X^* = 1 + X + X^2 + \ldots$ 、$+$ は集合の直和、$\times$ は集合の直積、1 は 1 個の元からなる集合を表します)

CLP^{0}のプログラムの定義

節は頭部と本体の組で、頭部は1個の項、本体は有限個(0個でもよい)の項からなります。節の論理的な意味は、頭部の項を h 、本体の項を $b_1, b_2, \ldots, b_n$ 、節に現れる変数を $v_1, v_2, \ldots, v_m$ とすると「任意の $v_1, v_2, \ldots, v_m$ に対して $b_1 \wedge b_2 \wedge \ldots \wedge b_n$ ならば h 」となります($\wedge$ は論理積を表します)。プログラムの論理的な意味は、節の論理積となります(節の集合 Q は $Q = T \times T^*$ 、プログラムの集合 P は $P = Q^*$ と書くことができます)。

$\mathbf{CLP^0}$のプログラムによる論理式の変換

ここでは論理式を次のような長方形の図で表します。

$$\begin{bmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ t_{21} & t_{22} & \dots & t_{2n} \\ \dots & \dots & \dots & \dots \\ t_{m1} & t_{m2} & \dots & t_{mn} \end{bmatrix}$$

各 t_{ij} は項またはtrueで、trueは「常に真である論理式」を表します(論理式の集合を E とおきます。$E = T^{**}$ と考えられます)。この記法の横の1行 $t_{i1}\ t_{i2}\ \dots\ t_{in}$ は $t_{i1}, t_{i2}, \dots, t_{in}$ の連言(合接、論理積)を表します(連言に現れる変数を $v_1, v_2, \dots, v_m$ とすると $t_{i1} \wedge t_{i2} \wedge \dots \wedge t_{in}$ を満たす $v_1, v_2, \dots, v_m$ が存在することを表します)。この記法全体は、それらの選言(離接、論理和)を表します。

ここではプログラム p を以下のような表記で表します。

$$p = \begin{bmatrix} h_1 & \mid & b_{11} & b_{12} & \dots & b_{1n} \\ h_2 & \mid & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ h_m & \mid & b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

ここで、横の1行は節を表し、h_i は頭部、$b_{i1}, b_{i2}, \dots, b_{in}$ は本体を表します。本体の個数が足りない行にはtrueを補って長方形になるようにします。

プログラム p は T から E への写像と考えることができます(ここでは以下の記法で表すことにします)。項 t に対して t と p が同じ変数を含まないように変数を変更した後、$t \otimes p$ を

$$t \otimes \begin{bmatrix} h_1 & \mid & b_{11} & b_{12} & \dots & b_{1n} \\ h_2 & \mid & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ h_m & \mid & b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} = \begin{bmatrix} \mathrm{equal}(t, h_1) & b_{11} & b_{12} & \dots & b_{1n} \\ \mathrm{equal}(t, h_2) & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \mathrm{equal}(t, h_m) & b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

と定義します。ここで $\mathrm{equal}(t, t')$ は、C または F に属するすべての f に対して(f のアリティ(項の個数)を n とすると)プログラムの中に「$\mathrm{equal}(x_1, y_1)$、$\mathrm{equal}(x_2, y_2)$、…、$\mathrm{equal}(x_n, y_n)$ ならば $\mathrm{equal}(f(x_1, x_2, \dots, x_n), f(y_1, y_2, \dots, y_n))$」の意味の節を含むような述語とします(Prologの記法では $\mathrm{equal}(f(x_1, x_2, \dots, x_n), f(y_1, y_2, \dots, y_n))\ \text{:-}\ \mathrm{equal}(x_1, y_1), \mathrm{equal}(x_2, y_2), \dots, \mathrm{equal}(x_n, y_n).$ という節)。この変換をここではズームインと呼ぶことにします。論理式の簡約を「定義の展開」、「代入」、「正規化(コンストラクタだけの式を正規形に変換)」に分けて考えます。「定義の展開」をズームインと呼ぶことにします。

これを拡張してプログラム p は E から E への写像と考えることができます。論理式 e に対するズームインは以下のように定義します。t と p が同じ変数を含まないように変数を変更した後、

$$e = \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1l} \\ t_{21} & t_{22} & \dots & t_{2l} \\ \dots & \dots & \dots & \dots \\ t_{k1} & t_{k2} & \dots & t_{kl} \end{bmatrix}, p = \begin{bmatrix} h_1 & \mid & b_{11} & b_{12} & \dots & b_{1n} \\ h_2 & \mid & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ h_m & \mid & b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

のとき

$$e \otimes p = \begin{bmatrix} t_{11} & t_{12} & \dots & t_{1l} \\ t_{21} & t_{22} & \dots & t_{2l} \\ \dots & \dots & \dots & \dots \\ t_{k1} & t_{k2} & \dots & t_{kl} \end{bmatrix} \otimes \begin{bmatrix} h_1 & \mid & b_{11} & b_{12} & \dots & b_{1n} \\ h_2 & \mid & b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ h_m & \mid & b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

は、まず t と p が同じ変数を含まないように p に現れる変数に (i, j) というインデックスをつけたものを p_{ij} として、以下のような図式を作ります。

$$\begin{bmatrix} t_{11} \otimes p_{11} & t_{12} \otimes p_{12} & \dots & t_{1l} \otimes p_{1l} \\ t_{21} \otimes p_{21} & t_{22} \otimes p_{21} & \dots & t_{2l} \otimes p_{2l} \\ \dots & \dots & \dots & \dots \\ t_{k1} \otimes p_{k1} & t_{k2} \otimes p_{k2} & \dots & t_{kl} \otimes p_{kl} \end{bmatrix}$$

この図式は、項からなる図式の項の代わりに論理式にして、これを論理式として展開して選言標準形にしたものを表すものとします($(a+b)(c+d) = ac+ad+bc+bd$ のような変換をする)。p によって n 回ズームインした $e \otimes p \otimes p \otimes \dots \otimes p$ (p が n 個)を $e \otimes p^n$ と書くことにします。このとき n 個めの p に現れる変数には (n, i, j) というインデックスをつけるとして、インデックスがつけられた変数の全体を改めて V とします。

S を V から T への写像全体の集合とします。S の元を代入と呼びます。代入はコンストラクタと関数を保存する T から T への写像と見ることもできます。これを拡張して代入はコンストラクタと関数を保存する E から E への写像と見ることもできます。

項 $t_1, t_2, \dots, t_m$ とプログラム p に対して e を $t_1, t_2, \dots, t_m$ の連言(これを $e = [t_1\ t_2\ \dots\ t_m]$ と書くことにします)とします。(e, p) の評価を、$s(e \otimes p^n) = \mathrm{true}$ を満たす代入 s と自然数 n が存在するとき (e, p) は成功、そうでないとき (e, p) は失敗とします。e に現れる変数を $v_1, v_2, \dots, v_k$ とすると、(e, p) が成功であることと、e を満たす $v_1, v_2, \dots, v_k$ が存在することが p から証明可能であることは同値になります。

プログラミング言語Prologの実行順序と構文

プログラミング言語Prologの実行順序は連言の順序に依存するものになっています。$e \otimes p^n$ の展開のときに左側の論理式から連言の順序を保存するように展開していくようにします。連言のすべての項がtrueとなるとき連言はtrue、連言がtrueではなく、かつその要素がすべてコンストラクタであるとき連言は失敗ということにします。$s(e \otimes p^n) = \mathrm{true}$ を満たす代入 s と自然数 n が存在し、かつ上記の連言の順序で最初にtrueとなる連言より前の連言はすべて失敗のとき成功、そうでないとき失敗とします。

プログラミング言語Prologの記法は以下のようになっています。変数、コンストラクタ、関数は英数字からなる文字列とします(これを名前と言います)。変数名は大文字から始まるものとします。コンストラクタ c から作られる項(アリティ n)は $c(t_1, t_2, \dots, t_n)$ のように書きます。$n=0$ のときは c と書きます。関数 f から作られる項(アリティ n)は $f(t_1, t_2, \dots, t_n)$ のように書きます。$n=0$ のときは f と書きます。

節は

h :- $b_1, b_2, \dots, b_n$. (本体の項の個数が1個以上のとき)

または

h . (本体の項の個数が0個のとき)

のように書きます。プログラムは節を並べた物となります。このプログラムに対して、頭部を持たない節

?- $b_1, b_2, \dots, b_n$.

が入力されると、この節とプログラムに対して上記の評価が行われます。

プログラミング言語Prologのプログラムは以下のようになります。

h_1 :- $b_{11}, b_{12}, \dots, b_{1n_1}$.

h_2 :- $b_{21}, b_{22}, \dots, b_{2n_2}$.

…………

h_m :- $b_{m1}, b_{m2}, \dots, b_{mn_m}$.

?- $g_1, g_2, \dots, g_k$.

ここで各 h_i 、b_{ij} 、g_i は、項となります。このPrologの記法は後でプログラムを表すために使うことがあります。

代入 s と s' に対して $s' = s''\bullet s$ を満たす代入 s'' が存在するとき、$s \leq s'$ と書くことにします。ここで $\bullet$ は写像の合成を表します。$s \leq s'$ かつ $s \neq s'$ のとき $s < s'$ と書きます。代入全体の集合 S は $\leq$ によって順序集合となります。

Prologのプログラムでの項 t の評価の実行手順は(まず変数が重複しないように変更した後) $s(t) = s(h_1)$ を満たす代入 s があれば、そのような s のうちで最小のもの(最も一般的なもの、これを s とします)をとって、$s(b_{11})$、$s(b_{12})$ の順に同様のことをやっていって、すべて成功であれば成功、そうでなければ h_2 に行って同じことをやっていきます。もしそのような s がなければ次に h_2、その次に h_3 のように同じことをやっていって、どこかで成功になれば結果は成功で終了、h_m まで行っても成功しなかったときは結果は失敗で終了となります。

CLPthの実行順序

ここではプログラミング言語Haskellで無限の長さのリスト(ストリーム)を扱うのと同様のことを実現する方法を考えます。Haskellは実行順序は不定なのですが、この場合は(暗黙的に)実行順序があります。この考え方はいろいろあると思いますが、ここでは構造の最初からの長さ(木の高さということにします)が確定するものを考えます。

連言 c と代入 s に対して、$s(c)$ がtrueではなく、かつその要素がすべてコンストラクタであるとき (c, s) は失敗ということにします。論理式 e と代入 s に対して、$s(e)$ のすべての連言が失敗であることを (e, s) は失敗ということにします。(e, s) が失敗ではないとき、(e, s) は解の候補であるということにします。

連言 $e = [t_1 t_2 \dots t_m]$ とプログラム p に対して代入の列 $s_0 \leq s_1 \leq s_2 \leq \dots$ が存在して、任意の自然数 k に対して $(e \otimes p^n, s_k)$ は解の候補で、$s_k(e)$ の先頭から k 個までの要素はすべてコンストラクタであるとき、(e, p) は木の高さに関して連続ということにし、このような (e, p) の全体をCLPth ということにします。

CLP^{1}の実行順序

ここではプログラミング言語Haskellでリストの先頭から何個目かまでの要素が確定するというものと同様のことを実現することを考えます。

B を C の部分集合で、B の元 b のアリティはすべて2であるものとします。連言 $e = [t_1 t_2 \dots t_m]$ とプログラム p に対して代入の列 $s_0 \leq s_1 \leq s_2 \leq \dots$ が存在して、任意の自然数 k に対して $(e \otimes p^n, s_k)$ は解の候補で、e に含まれる任意の変数 v に対して $s_k(v)$ はすべてコンストラクタからなる項であるか、または $b_0(x_0, b_1(x_1, b_2(x_2, \dots)))$ という形で $b_0, b_1, b_2, \dots$ は B の元で $x_0, x_1, x_2, \dots, x_k$ はすべてコンストラクタからなる項であるとき、(e, p) はストリームに関して連続、このような (e, p) の全体をCLP^1 ということにします。

CLP^{2}の実行順序

前節のCLP^{1}は1次元のリストに関するものでしたが、これを2次元に拡張することを考えます。

B_1, B_2 を C の部分集合で、B_1, B_2 の元のアリティはすべて2であるものとします。連言 $e = [t_1 t_2 \dots t_m]$ とプログラム p に対して代入の列 $s_{00} \leq s_{01} \leq s_{02} \leq \dots s_{10} \leq s_{11} \leq s_{12} \leq \dots s_{20} \leq s_{21} \leq s_{22} \leq \dots$ が存在して、任意の自然数 k, l に対して $(e \otimes p^n, s_{kl})$ はの解の候補で、e に含まれる任意の変数 v に対して $s_{kl}(v)$ はすべてコンストラクタからなる項であるか、または $b_0(x_0, b_1(x_1, b_2(x_2, \dots)))$ という形で $b_0, b_1, b_2, \dots$ は B_1 の元で $x_0, x_1, x_2, \dots, x_{k-1}$ はすべてコンストラクタからなる項で $x_k = c_0(y_0, c_1(y_1, c_2(y_2, \dots)))$ という形で $c_0, c_1, c_2, \dots$ は B_2 の元で $y_0, y_1, y_2, \dots, y_l$ はすべてコンストラクタからなる項であるとき、(e, p) は2次元のストリームに関して連続、このような (e, p) の全体をCLP^2 ということにします。

CLP$^+$の実行順序

プログラミング言語C#では、関数の中でyield return文が使われて、その関数がforeach文から呼び出されるかその関数から作られたあるオブジェクトのMove Next()メソッドが呼び出されると、関数の呼び出しはyield returnのところでいったん呼び出し元に制御が戻って、再び呼び出されるとyield returnの次から再開されるという仕組みがあります(コルーチンと呼ばれるようです)。この動作は、yield return文に来たときに中断した場所とそのときの変数の値などの状態を記憶しておいて、再開されたときはその場所と状態から始めると考えることができます。

また、複数の関数があって、呼び出しごとに異なる関数が呼び出されると考えることもできます。yield return文でふつうにリターンする関数と、yield return文では何もせずに次に進む関数の2つの関数があって、1回目の呼び出しではリターンする関数、2回目の呼び出しでは何もしない関数が呼び出されると考えることもできます。

ここではこの考え方で論理プログラミング言語にyield return文と同様なものを取り入れることを試みます。これはプログラムの実行中に一時的に実行を中断して、状態を見たり状態を変更したりできるもので、ここではブレークポイントと呼ぶことにします。

順序集合(ここでは半順序集合の意味で使います)の列 X_i $(i=1, 2, 3, \ldots)$ に対して直積 $\prod_{i=1}^{\infty} X_i$ を集合 $\prod_{i=1}^{\infty} X_i$ に辞書式順序を定義したものとします。

次のようなプログラムを考えます。

$$p^+ = \begin{bmatrix} h_1 & | & b_{11} & b_{12} & \ldots & b_{1n} \\ h_2 & | & b_{21} & b_{22} & \ldots & b_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ h_m & | & b_{m1} & b_{m2} & \ldots & b_{mn} \end{bmatrix}$$

ただしここで各 b_{ij} はブレークポイントが設定された項と設定されていない項の2種類があります。T^+ をブレークポイントが設定された項全体の集合とします。T^+ は T と同じものです。各 b_{ij} は T または T^+ の元となります。このようなプログラム p^+ 全体の集合を P^+ とします。$P^+ = (Q^+)^*$ 、$Q^+ = T \times (T + T^+)^*$ となります。

b_{ij} が T^+ の元であるもの全体を $\{t_1^+, t_2^+, \ldots, t_k^+\}$ とおき、t_i^+ に対応する T の元を t_i とおきます。p^+ の t_i^+ を t_i に置き換えて、他の t_j^+ をtrueに置き換えたものを p_i とおきます。p^+ の t_j^+ をすべてtrueに置き換えたものを p_0 とおきます。

$X = \{x, y_1, y_2, \ldots, y_k\}$ とおきます。X は順序集合で、各 i に対して $x \leq y_i$ を満たすとします。$BP_\mu = \prod_{i=1}^{\mu} X_i$ とおきます。BP_μ をブレークポイントと呼ぶことにします。BP_μ の 元 z に $BP_\mu \times X$ の部分集合 $\{(z, x), (z, y_1), (z, y_2), \ldots, (z, y_k)\}$ を対応させる写像をブレークポイントの分割ということにします。z に (z, x) を対応させることによって $BP_\mu \subseteq BP_\mu \times X$ と考えたときの $BP = \cup_{i=1}^{\infty} X_i$ もブレークポイントということにします。

e^+ を BP_μ から論理式全体の集合 E への写像とすると $e^+ \otimes p^+$ は $BP_{\mu+1}$ から E への写像で

$(e^+ \otimes p^+)(z, x) = e^+(z) \otimes p_0$

$(e^+ \otimes p^+)(z, y_i) = e^+(z) \otimes p_i \ (\forall i)$

を満たすものとします。

BP の元は時刻に対応していて、ある時刻に1つの BP の元が指定されてそれに対応するプログラムが呼び出されるシステムがあれば、「イベント駆動型」のシステムができると考えられます。

連言 $e = [t_1 t_2 \ldots t_m]$ に対して BP から S への順序を保存する写像 σ が存在して、任意の BP の元 x に対して $((e \otimes (p^+)^n)(x), \sigma(x))$ が解の候補であるとき、(e, p^+) はブレークポイントに関して連続、このような (e, p^+) の全

体をCLP^+ということにします。

結論

この文章では無限オブジェクトに関する実行順序CLP^{th}、CLP^1、CLP^2、CLP^+を定義しました。Haskellの無限の長さのストリームに関する実行順序と同様の実行順序を持つものがCLP^1、オブジェクト指向言語の永久に存在するオブジェクトに関する実行順序と同様の実行順序を持つものがCLP^+で、CLP^1とCLP^+は異なるということを示すことが目標ですが、現在はできていません。

参考文献

Prolog
お気楽 Prolog プログラミング入門
http://www.geocities.jp/m_hiroi/prolog/

GHC
並行論理プログラミング言語 GHC / KL1
http://www.ueda.info.waseda.ac.jp/~ueda/readings/GHC-intro.pdf

Haskell
本物のプログラマはHaskellを使う
http://itpro.nikkeibp.co.jp/article/COLUMN/20060915/248215/

F#
F#入門
http://winterradish.web.fc2.com/

C#
C# によるプログラミング入門
http://ufcpp.net/study/csharp/

Java
java.com
http://www.java.com/ja/

C++
C++ Language Tutorial
http://www.cplusplus.com/doc/tutorial/

Smalltalk
Smalltalk入門
http://www.oklab.org/program/gnu_smalltalk.html

履歴

2013年5月28日 Revision 1.00