

フラクタル代数の極限フラクタルオブジェクト

2015年1月7日 Revision 2.00

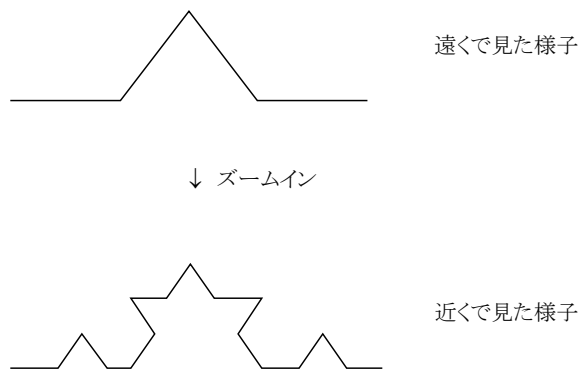
古田秀和 (FURUTA Hidekazu)

背景と目的

論理プログラミング言語(Prologのような)でオブジェクト指向のオブジェクトを表すときに、任意の無限長のデータを表すということが1つの問題になっていると考えられます。

実行順序を考えないPrologの仕組み(これを数理的論理プログラミングと呼ぶことにします)を考えます。数理的論理プログラミングのデータとなっている論理式全体の集合 E はブール代数となっています。「プログラムを論理式に一度だけ施す」という「作用」の合成と、「選択」(プログラムの「節」を選択することを表します)によって、プログラム全体の集合 P は(E への作用も含めて)べき等半環となります。

論理式とプログラムと論理式の論理積と論理和、ズームインと選択という演算の組を(以前フラクタル図形の描画のため用いた演算の体系と似たものになるということで)「フラクタル代数」と呼ぶことにします。



数理的論理プログラミングで生成される無限の長さの再帰的なデータは、論理式 e とプログラム p が定まると代入が定まるもので、これを一般フラクタルオブジェクトと呼ぶことにします。

これに対してユーザーからの入力のような任意のデータの列は、論理式 e とプログラムの列 p_1, p_2, \dots の「極限」から代入が定まる(有理数の列の極限として実数が定まるように)と考えられます。これを極限フラクタルオブジェクトと呼ぶことにします。

任意のデータの列を表すことができるということはオブジェクト指向のオブジェクトの一つの性質と考えられます。この文章ではオブジェクトを数理的論理プログラミングで表すための体系を考えていきます。

数理的論理プログラミング言語MLPの項の定義

プログラミング言語 Prolog の実行順序を考えないものを定義します。これを数理的論理プログラミング言語 MLP と呼ぶことにします(または純粋論理プログラミング言語)。

集合 V 、 C 、 J を考えます。 V は変数の集合(可算)、 C はコンストラクタ(定数も含む)の集合(有限)、 J は述語の集合を表すものとします。各コンストラクタ c および述語 j には、アリティ(構造の項の個数)を表す自然数(0以上)が対応しています。

「構造の項」は変数、「コンストラクタの項」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「構造の項」の組と再帰的に定義されます。「述語の項」は述語(アリティを n とする)と n 個の「構造の項」の組、と再帰的に定義されます。「述語の項」を項と呼ぶことにします。

項の集合 T は $T = J \times K^*$ 、 $K = V + C \times K^*$ とすることもできます (K は構造の項を表します)。ここで集合 X に対して $X^n = X \times X \times \dots \times X$ (n 個)、 $X^* = 1 + X + X^2 + \dots$ 、 $+$ 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。

MLP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, \dots, b_n 、節に現れる変数を v_1, v_2, \dots, v_m とすると「任意の v_1, v_2, \dots, v_m に対して $b_1 \wedge b_2 \wedge \dots \wedge b_n$ 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)。

MLP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。

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

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

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

と定義します。ここで $t \equiv t'$ 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「 $x_1 \equiv y_1, x_2 \equiv y_2, \dots, x_n \equiv y_n$ ならば $f(x_1, x_2, \dots, x_n) \equiv f(y_1, y_2, \dots, y_n)$ 」の意味の節を含むような述語とします (Prolog の記法では

$$f(x_1, x_2, \dots, x_n) \equiv f(y_1, y_2, \dots, y_n) :- x_1 \equiv y_1, x_2 \equiv y_2, \dots, x_n \equiv y_n.$$

という節)。この演算 \otimes 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

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、

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

のとき

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

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。

この変数のインデックスを $\eta(n, i_n, j_n)$ と書くことにします。これを組み合わせた

$$\eta(n, i_1, j_1) \otimes \dots \otimes \eta(n, i_n, j_n)$$

をズームインパスと呼ぶことにします。元の変数を v とすると、ズームインパスは新しい変数名につける接頭語のようなもので、ズームインパスを頭につけて、新しい変数名を

$$\eta(n, i_1, j_1) \otimes \dots \otimes \eta(n, i_n, j_n) \otimes v$$

とすることにより、変数名が重複しないようにすることができます。

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。同様に、代入はコンストラクタと関数を保存する P から P への写像と見ることもできます。

代入 σ と τ に対して $\sigma \bullet \tau$ を $(\sigma \bullet \tau)(e) = \sigma(\tau(e))$ ($e \in E$)、 $(\sigma \bullet \tau)(p) = \sigma(\tau(p))$ ($p \in P$) と定義します。代入 σ と τ に対して $\rho \bullet \sigma = \tau$ を満たす代入 ρ が存在するとき $\sigma \leq \tau$ と定義します。 $\sigma \leq \tau$ で $\sigma \neq \tau$ のとき $\sigma < \tau$ と定義します。

論理式の横の1行を「行」と呼ぶことにします。 $t \equiv t'$ は (Prologの記法で) t と t' を書いたとき一致するならば true、一致しないならば false とします。論理式のどれか1つの行が true であるときその論理式は true とします。

論理式 e とプログラム p に対して、 $s(e \otimes p^n) = \text{true}$ 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の実行順序は論理式の行の順序に依存するものになっています。Prologでは行が **true, true, ..., true, false, ...**

となっているときその行は **false** とします。論理式のどれか1つの行が **true** で、その前のすべての行が **false** であるときその論理式は **true** 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。

$h_1 :- b_{11}, b_{12}, \dots, b_{1m_1}.$

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

.....

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

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

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 $s' = s'' \circ s$ 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 \circ 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 に行き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。

項 t, t' に対し、 $s(t) = s(t')$ となる代入 s を単一化代入(unifier)といいます。 $s(t) = s(t')$ となる代入 s が存在するとき、項 t, t' は単一化可能(unifiable)であるといいます。代入 s が2つの項 t, t' の単一化代入で、任意の t, t' の単一化代入 s' に対して、 $s' = s'' \circ s$ となる代入 s'' が存在するとき、 s は最も一般的な単一化代入(most general unifier, mgu)であるといいます。

MLPのプログラムの(Prologの記法では)

$f(x_1, x_2, \dots, x_n) \equiv f(y_1, y_2, \dots, y_n) :- x_1 \equiv y_1, x_2 \equiv y_2, \dots, x_n \equiv y_n.$

に相当する部分を単一化作用と呼ぶことにします。単一化作用を何回か繰り返すことと代入によって \equiv を含まない行にすることができます。このときの代入を単一化代入と呼ぶことにします。このときの単一化作用の何回かの繰り返しはプログラムの作用のときに毎回行われると考えて、単一化代入によって行には \equiv を含まなくなると考えることにします。

MLPの論理式のブール代数

MLPの論理式全体の集合 E はブール代数になっています。すなわち E の元 T と F が存在し、任意の E の元 x に対して

$\neg x$ が存在し、二項演算論理積 \wedge と論理和 \vee が存在して任意の E の元 x, y, z に対して以下の規則が成り立ちます。

$$(x \wedge y) \wedge z = x \wedge (y \wedge z) \quad (\text{B1: 論理積の結合法則})$$

$$x \wedge x = x \quad (\text{B2: 論理積のべき等法則})$$

$$x \wedge T = T \wedge x = x \quad (\text{B3})$$

$$x \wedge F = F \wedge x = F \quad (\text{B4})$$

$$(x \vee y) \vee z = x \vee (y \vee z) \quad (\text{B5: 論理和の結合法則})$$

$$x \vee x = x \quad (\text{B6: 論理和のべき等法則})$$

$$x \vee F = F \vee x = x \quad (\text{B7})$$

$$x \vee T = T \vee x = T \quad (\text{B8})$$

$$(x \vee y) \wedge x = x \wedge (x \vee y) = x, (x \wedge y) \vee x = x \vee (x \wedge y) = x \quad (\text{B9: 吸収法則})$$

$$(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z), x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \quad (\text{B10: 分配法則1})$$

$$(x \wedge y) \vee z = (x \wedge z) \vee (y \wedge z), x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \quad (\text{B11: 分配法則2})$$

$$x \wedge y = y \wedge x \quad (\text{B12: 論理積の交換法則})$$

$$x \vee y = y \vee x \quad (\text{B13: 論理和の交換法則})$$

$$x \wedge \neg x = F \quad (\text{B14})$$

$$x \vee \neg x = T \quad (\text{B15})$$

以下の議論では、このうち「任意の E の元 x に対して $\neg x$ が存在する」という条件と(B14)、(B15)は必要ありません(こうすると最大元、最小元を持つ分配束となります)。

MLPのプログラムのべき等半環

MLPのプログラム全体の集合 P はべき等半環(idempotent semiring, dioid)になっています。すなわち P の元 1 と 0 が存在し、二項演算ズームイン \otimes と選択 \vee が存在して任意の P の元 p, q, r に対して以下の規則が成り立ちます。

$$(p \otimes q) \otimes r = p \otimes (q \otimes r) \quad (\text{D1: ズームインの結合法則})$$

$$p \otimes 1 = p \quad (\text{D2: ズームインの単位元})$$

$$p \otimes 0 = 0 \quad (\text{D3})$$

$$(p \vee q) \vee r = p \vee (q \vee r) \quad (\text{D4: 選択の結合法則})$$

$$p \vee p = p \quad (\text{D5: 選択のべき等法則})$$

$$p \vee 0 = p \quad (\text{D6: 選択の単位元})$$

$$(p \vee q) \otimes r = (p \otimes r) \vee (q \otimes r), p \otimes (q \wedge r) = (p \otimes q) \vee (p \otimes r) \quad (\text{D7: 分配法則})$$

$$p \vee q = q \vee p \quad (\text{D8: 選択の交換法則})$$

MLPのプログラムの論理式への作用

MLPのプログラム p の作用 $\otimes p$ は論理積、論理和、 T 、 F を保存する E から E への写像で任意の論理式 x, y とプログラム p, q に対して以下の条件を満たします。

$$(x \otimes p) \otimes q = x \otimes (p \otimes q) \quad (\text{P1})$$

$$(x \wedge y) \otimes p = (x \otimes p) \wedge (y \otimes p) \quad (\text{P2: 論理積の保存})$$

$$(x \vee y) \otimes p = (x \otimes p) \vee (y \otimes p) \quad (\text{P3: 論理和の保存})$$

$$T \otimes p = T \quad (\text{P4: } T \text{の保存})$$

$$F \otimes p = F \quad (\text{P5: } F \text{の保存})$$

$$x \otimes 1 = x \quad (\text{P6})$$

$$x \otimes 0 = F \quad (\text{P7})$$

フラクタル代数の定義

集合 E と P に上記の演算(論理積、論理和、ズームイン、選択)が定義されて上記の条件(B1)–(B15)、(D1)–(D8)、

(P1)–(P7)を満たすとき $(E, \wedge, \vee), (P, \otimes, \vee), \otimes$ の組をフラクタル代数と呼ぶことにします(以前フラクタルを記述するときに使った演算の体系と同様のものであるためこのように呼ぶことにします)。

Prologの実行順序を表すために、フラクタル代数の定義から(B12: 論理積の交換法則)、(B13: 論理和の交換法則)、(D8: 選択の交換法則)を除いたものを考えます。これをPフラクタル代数と呼ぶことにします。

一般フラクタルオブジェクト

代入 σ は、論理積、論理和、T、Fを保存する E から E への写像で、「構造」を保存し、 $\sigma^2 = \sigma$ を満たすものと考えられます。論理式 e とプログラム p に対して、 $e \otimes p^n$ の単一化代入 u_n に対して $u_n \leq \theta_n$ を満たす代入の列 $\theta_1 \leq \theta_2 \leq \dots$ が存在して、任意の自然数 n に対して $(s \bullet \theta_n) (e \otimes p^n)$ の1つの行が true となるような代入 s が存在するとします。さらに $\theta_1, \theta_2, \dots$ が最も一般的な代入であるとき、 e に現れる1つの変数 v に対して $\theta_1(v), \theta_2(v), \dots$ という列を、 (e, p) で定まる一般フラクタルオブジェクト(の1つ)と呼ぶことにします。

プログラムが一般フラクタルオブジェクトを表すと考えることもできます。これは次の節で考えます。

極限フラクタルオブジェクト

ユーザーからの入力のような任意のデータの列を表すことを考えます。任意のデータの列を表すことができるということはオブジェクト指向のオブジェクトの一つの性質と考えられます。

論理式 e とプログラム p と、 e に現れる1つの変数 v に対して、 $\theta_1(v), \theta_2(v), \dots$ を、 (e, p) で定まる一般フラクタルオブジェクトとします。代入の列 $\sigma_1 \leq \sigma_2 \leq \dots$ を各 σ_i が $\theta_i \leq \sigma_i$ を満たすものとします。 $\sigma_1(v), \sigma_2(v), \dots$ という列を、極限フラクタルオブジェクトと呼ぶことにします。極限フラクタルオブジェクトを定めるプログラムの列を考えます。 $\sigma_i = \delta_i \bullet \theta_i$ において δ_i に対応する項を p の各行に追加したものを p_i とおきます。 σ_i は $e \otimes p_1 \otimes p_2 \otimes \dots \otimes p_i$ のある行が true で、 p_i の単一化代入より大きい代入の中で最も一般的なものとなります。

$p_1, p_1 \otimes p_2, p_1 \otimes p_2 \otimes p_3, \dots$

という列を $[\otimes p_i]$ と書くことにします。 $\sigma_1(v), \sigma_2(v), \dots$ という列を、 $(e, [\otimes p_i])$ で定まる極限フラクタルオブジェクト(の1つ)と呼ぶことにします。

この議論は逆にプログラムの方から決めていくこともできます。式 e とプログラムの列 p_1, p_2, p_3, \dots に対して

$e \otimes p_1, e \otimes p_1 \otimes p_2, e \otimes p_1 \otimes p_2 \otimes p_3, \dots$

という列を $e \otimes [\otimes p_i]$ と書くことにします。上で述べたようにこの列によって極限フラクタルオブジェクトが定まります。

集合 X の元の n 個の列全体の集合を X^n とすると、 $m \geq n$ のときに $f_{mn}: X^m \rightarrow X^n$ を最初の n 個の項をとる写像とすると、集合 X の元の可算個の列全体の集合 X^ω は X^n と f_{mn} からなる射影系の射影的極限となります。 $\pi_n: X^\omega \rightarrow X^n$ を X^ω の元 $[x_1, x_2, \dots]$ に X^n の元 $[x_1, x_2, \dots, x_n]$ を対応させる写像とします。 X^ω は X^n の直積 $\prod_n X^n$ の部分集合で $m \geq n$ のとき X^m の成分を $[x_1, x_2, \dots, x_m]$ とすると X^n の成分は $[x_1, x_2, \dots, x_n]$ となるような元の全体となります。

$e \otimes [\otimes p_i]$ は E^n と $f_{mn}: E^m \rightarrow E^n$ ($m \geq n$) からなる射影系の射影的極限 E^ω の元と考えることができます。 $e \otimes [\otimes p_i]$ によって極限フラクタルオブジェクトが定まることから、 $e \otimes [\otimes p_i]$ 自体を極限フラクタルオブジェクトと呼ぶことにします。

一般フラクタルオブジェクトは、 $e \otimes [p]$ という極限フラクタルオブジェクトと考えることもできます(極限フラクタルオブジェクトの p_i がすべて p であるもの)。

極限フラクタルオブジェクト $e \otimes [\otimes p_i]$ の p_i がある実数の小数点以下 i 桁目の数だとします(たとえばユーザーからの入力とするとユーザーがその数を入力したものとします)。このような極限フラクタルオブジェクトは少なくとも実数の濃度だけ存在

しより。

一般フラクタルオブジェクトは(ここでは議論しませんが、前半の議論と同様に)有限個のコンストラクタと可算個の変数によって生成された「構造の項」と有限個の述語によって構成される項によって生成されたフラクタル代数の元と考えられるので可算個と考えられます。

今後の展望

有理数の列の極限として実数が定まるように、極限フラクタルオブジェクトは一般フラクタルオブジェクトの極限と考えることができます。有理数と実数では濃度が異なるように、一般フラクタルオブジェクトと極限フラクタルオブジェクトでは濃度が異なると考えられます。この議論をするためにはフラクタル代数を生成する元の定義が必要となります。

任意のデータの列を表すことができるということはオブジェクト指向のオブジェクトの一つの性質と考えられます。数理的論理プログラミングで極限フラクタルオブジェクトを扱うことができるようにするため、関数プログラミングでの無限長のデータの扱い方を参考にして機能を追加したいと考えています。このようなデータを扱うためのいろいろな関数があり、相互に関連していますが、それらに共通のもの(極限の最大公約数のようなものと考えられます)を考えるというのが1つの方法になると思われます。

数理的論理プログラミングの各項で1つの変数に注目することとして論理式を非可換にすることで実行順序を表すことにより、関数プログラミング風にすることができます(変数だけに注目して変数に順序を導入すれば良いのかもしれませんが)。関数の合成で実行順序を表すことによって数学的な関数の理論が適用できるようになると思われます。関数プログラミングにおける構造とオブジェクト指向のオブジェクトの関係を調べていくことができると思われます。

参考文献

Dioid Algebra

<http://www.dca.fee.unicamp.br/~madaca/dioidAlgebra.html>

べき等半環

檜山正幸のキマイラ飼育記

<http://d.hatena.ne.jp/m-hiyama/20141110/1415574862>

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/>

履歴

2014年12月28日 Revision 1.00

2015年1月7日 Revision 2.00