フラクタル代数の極限フラクタルオブジェクト

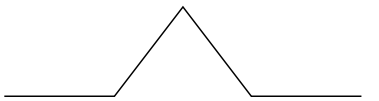
2014年12月28日 Revision 1.00

古田秀和 (FURUTA Hidekazu)

この文章の目的

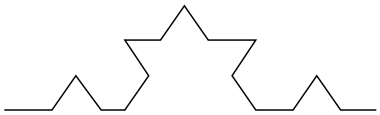
実行順序を考えないPrologの仕組み(これを数理的論理プログラミングと呼ぶことにします)を考えます。数理的論理プログラミングのデータとなっている論理式全体の集合 E はブール代数となっています。「プログラムを論理式に一度だけ施す」という「作用」の合成と、「選択」(プログラムの「節」を選択することを表します)によって、プログラム全体の集合 P は(E への作用も含めて)べき等半環となります。

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。



遠くで見た様子

↓ ズームイン



近くで見た様子

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 $p_1, p_2, \ldots$ 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 、F 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)、F は関数の集合を表すものとします。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、×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 $b_1, b_2, \ldots, b_n$、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（∧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。

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

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 $t_{i1}\, t_{i2} \ldots t_{in}$ は $t_{i1}, t_{i2}, \ldots, t_{in}$ の連言（合接、論理積）を表します（連言に現れる変数を $v_1, v_2, \ldots, v_m$ とすると $t_{i1} \wedge t_{i2} \wedge \ldots \wedge t_{in}$ を満たす $v_1, v_2, \ldots, v_m$ 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。

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

ここで、横の1行は節を表し、h_i は頭部、$b_{i1}, b_{i2}, \ldots, b_{in}$ 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

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

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「$x_1 \equiv y_1$、$x_2 \equiv y_2$、…、$x_n \equiv y_n$ ならば $f(x_1, x_2, \ldots, x_n) \equiv f(y_1, y_2, \ldots, y_n)$」の意味の節を含むような述語とします(Prologの記法では

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

という節)。この演算 ⊗ 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、

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

のとき

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

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。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、$s(e\otimes p^n)=$ 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 :- $b_1, b_2, \ldots, b_n$. (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
?- $b_1, b_2, \ldots, b_n$.
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 :- $b_{11}, b_{12}, \ldots, b_{1n_1}$.
h_2 :- $b_{21}, b_{22}, \ldots, b_{2n_2}$.
.....
h_m :- $b_{m1}, b_{m2}, \ldots, b_{mn_m}$.
?- $g_1, g_2, \ldots, g_k$.
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。
$(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (B1: 論理積の結合法則)
$x \wedge y = y \wedge x$ (B2: 論理積の交換法則)
$x \wedge x = x$ (B3: 論理積のべき等法則)
$x \wedge \neg x = \mathrm{F}$ (B4)
$(x \vee y) \vee z = x \vee (y \vee z)$ (B5: 論理和の結合法則)
$x \vee y = y \vee x$ (B6: 論理和の交換法則)
$x \vee x = x$ (B7: 論理和のべき等法則)
$x \vee \neg x = \mathrm{T}$ (B8)
$(x \vee y) \wedge x = (x \wedge y) \vee x = x$ (B9: 吸収法則)
$(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z), x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ (B10: 分配法則1)
$(x \wedge y) \vee z = (x \wedge z) \vee (y \wedge z), x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ (B11: 分配法則2)

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⊗と選択∨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。
$(p \otimes q) \otimes r = p \otimes (q \otimes r)$（D1: ズームインの結合法則）
$p \otimes 1 = p$（D2: ズームインの単位元）
$p \otimes 0 = 0$（D3）
$(p \vee q) \vee r = p \vee (q \vee r)$（D4: 選択の結合法則）
$p \vee q = q \vee p$（D5: 選択の交換法則）
$p \vee p = p$（D6: 選択のべき等法則）
$p \vee 0 = p$（D7: 選択の単位元）
$(p \vee q) \otimes r = (p \otimes r) \vee (q \otimes r), p \otimes (q \wedge r) = (p \otimes q) \vee (p \otimes r)$（D8: 分配法則）

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。
$(x \otimes p) \otimes q = x \otimes (p \otimes q)$（P1）
$(x \wedge y) \otimes p = (x \otimes p) \wedge (y \otimes p)$（P2）
$(x \vee y) \otimes p = (x \otimes p) \vee (y \otimes p)$（P3）
$\mathrm{T} \otimes p = \mathrm{T}$（P4）
$\mathrm{F} \otimes p = \mathrm{F}$（P5）
$x \otimes 1 = x$（P6）
$x \otimes 0 = \mathrm{F}$（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)-(B11)、(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）。

Prologの実行順序を表すために、フラクタル代数の定義から(B2: 論理積の交換法則)、(B6: 論理和の交換法則)、(D5: 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

MLPのプログラムによる論理式の変換の$t \equiv h_i$の部分を単一化項と呼ぶことにします。

論理式eとプログラムpに対して、代入の列$\theta_1 \le \theta_2 \le \ldots$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で、その行の単一化項がすべてtrueとなるような代入sが存在するとします。さらに$\theta_1, \theta_2, \ldots$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$\theta_1(v), \theta_2(v), \ldots$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。

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、$\theta_1(v), \theta_2(v), \ldots$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$\sigma_1 \le \sigma_2 \le \ldots$を各$\sigma_i$が$\theta_i \le \sigma_i$を満たすものとします。$\sigma_1(v), \sigma_2(v), \ldots$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。$\sigma_i = \delta_i \bullet \theta_i$とおいて$\delta_i$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。$\sigma_i$は$e \otimes p_1 \otimes p_2 \otimes \ldots \otimes p_i, \ldots$のある行がtrueで、その

行の単一化項がすべて true となるような代入の中で最も一般的なものとなります。

$p_1, p_1 \otimes p_2, p_1 \otimes p_2 \otimes p_3, \ldots$

という列を $[\otimes p_i]$ と書くことにします。$\sigma_1(v), \sigma_2(v), \ldots$ 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。プログラムの列 $p_1, p_2, p_3, \ldots$ に対して

$p_1, p_1 \otimes p_2, p_1 \otimes p_2 \otimes p_3, \ldots$

という列を $[\otimes p_i]$ と書くことにします。$e \otimes p_1 \otimes p_2 \otimes \ldots \otimes p_i, \ldots$ のある行が true で、その行の単一化項がすべて true となるような代入の中で最も一般的なものを σ_i とおきます。$\sigma_1(v), \sigma_2(v), \ldots$ 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。一般フラクタルオブジェクトは、$(e, [\otimes p])$ で定まる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）。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。

今後の展望

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。この問題を扱うためには極限の定義を厳密にしていかなければならないと考えられます。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だけでは、オブジェクト指向のオブジェクトの一つの性質を扱うことはできないと考えられます。

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、追加すべき機能は何かを考えていく必要があります。

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)。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り、純粋関数プログラミングとオブジェクト指向のオブジェクトの関係も明らかになると思われます。

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