論理プログラミングによる構造を表す関数

2014年1月27日 Revision 1.00

古田秀和 (FURUTA Hidekazu)

背景

オブジェクト指向言語を特徴づけるものとして、(Microsoft .NET Framework のジェネリックコレクションのような)コレクション(たとえばList、Dictionary)を容易に扱うことができることが考えられます。このようなコレクションのクラスがオブジェクト指向のライブラリにあって、オブジェクト指向言語から使えるようになっています。オブジェクト指向のプログラミングに現れるものの中で、木のような再帰的な構造は再帰的なプログラムのプログラム上の位置を指定することによって実現されると考えることができます。

関数プログラミングではそのような構造は関数で定義されています。関数プログラミングでオブジェクト指向のライブラリを利用するため、構造を定義する方法を考察します。

H/Lシステム

Vを可算集合、C、Fを有限集合とします(Vの元を変数、Cの元を定数、Fの元を関数と呼びます)。$T = V + C \times T^* + F \times T^*$と再帰的に$T$を定義します($T$の元を項と呼びます。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)。$K = (T \backslash V)^*$とおきます(これは項の論理積を表します)。$Q = (T \backslash V) \times K$とおきます($Q$の元を節と呼び、節の最初の部分を頭部、2番目の部分を本体と呼びます)。$P = Q^*$とおきます(Pの元はQの元の論理積を表し、これを論理プログラムと呼ぶことにします)。

$E = K^*$とおきます(Eの元はKの元の論理和を表し、これを論理式と呼ぶことにします)。$s : V \to T$をC、Fの元を保存するように$s : E \to E$に拡張することができます。このような定数と関数を保存する$s : E \to E$を代入と呼び、代入全体の集合をSとおきます。

$t \in T$と$q \in Q$をとります。qの変数をtに含まれないものに置き換えたものをq'とおき、q'の頭部をh、q'の本体を$[b_1, b_2, \ldots, b_n]$とおきます($h, b_i \in T \backslash V \ (\forall i)$)。ここで括弧[]は論理積の$n$個の組を表すために使うことにします。$t \otimes q = [\mathrm{equal}(t, h), b_1, b_2, \ldots, b_n] \in K$と定義します。$\mathrm{equal}(t, h)$は$t$と$h$が一致することを表します。

$p = [q_1, q_2, \ldots, q_m] \in P$に対して$t \otimes p = (t \otimes q_1, t \otimes q_2, \ldots, t \otimes q_m) \in E$と定義します。ここで括弧()は論理和の$m$個の組を表すために使うことにします。この演算⊗によって$p : T \to E$とみなすことができます。$e = (k_1, k_2, \ldots, k_m) \in E$、$k_i = [t_1, t_2, \ldots, t_n] \in K$に対して、$k_i \otimes p = [t_1 \otimes p, t_2 \otimes p, \ldots, t_n \otimes p]$、$e \otimes p = (k_1 \otimes p, k_2 \otimes p, \ldots, k_m \otimes p)$を選言標準形に変形したものと定義します。この演算⊗によって$p : E \to E$とみなすことができます。$t \otimes p \otimes \ldots \otimes p$($p$が$n$個)を$t \otimes p^n$と書くことにします。

$k = [t_1, t_2, \ldots, t_n] \in K$と$p \in P$に対して、$s(k \otimes p^n)$のどれかの連言が空となる代入$s$と自然数$n$が存在するとき$(k, p)$は成功、そうでないとき$(k, p)$は失敗とします。

以下では評価の順序について考えます。項$t = (f, (t_1, t_2, \ldots, t_n))$を$t = f(t_1, t_2, \ldots, t_n)$と書きます。$t_1, t_2, \ldots, t_n$の1つが変数のとき、その変数を出力変数、項$t$を出力変数をもつ項ということにします。節$q = (h, (t_1, t_2, \ldots, t_n))$を$h : -t_1, t_2, \ldots, t_n$と書きます。

h と $t_1, t_2, \ldots, t_n$ が出力変数をもつ項で、h と t_n の出力変数が一致するとき節 q を出力変数をもつ節ということにします。以下では $k = [t_1, t_2, \ldots, t_n] \in K$ のすべての項は出力変数をもつ項、$p \in P$ のすべての節は出力変数をもつ節の場合を考えます。これをH/Lシステムと呼ぶことにします。（論理プログラミングによってHaskellのような純粋関数型言語を実現するという意味でこのように呼ぶことにします。）(k, p) が成功となる代入 s と自然数 n が存在するとき、t_n の出力変数 v の s による値 $s(v)$ を評価の結果と呼ぶことにします。

評価空間

以下では評価の順序について考えます。$k = [t_1, t_2, \ldots, t_n] \in K$ に対して $t_1, t_2, \ldots, t_n$ の出力変数をそれぞれ $v_1, v_2, \ldots, v_n$ とおきます。代入の列 $s_1 \leq s_2 \leq \ldots \leq s_n$ が存在して、任意の i に対して $s_i(v_i)$ は定数だけの式になっているものとします。この代入の列によって評価の順序が決まっていると考えます。

評価には時間がかかるものとして、ある項の評価が行われる時間の範囲を評価期間と呼ぶことにします。$t_1, t_2, \ldots, t_n$ の評価期間をそれぞれ $z_1, z_2, \ldots, z_n$ とおきます。全体の評価期間を z として $z = z_1 \gg z_2 \gg \ldots \gg z_n$ と表すことにします。これは全体の評価期間が $z_1, z_2, \ldots, z_n$ に分割されて、時間的な順序は $z_1, z_2, \ldots, z_n$ という順になっていることを表します。このとき評価期間には $z_1 \leq z_2 \leq \ldots \leq z_n$ という順序 $\leq$ があるものとします。

$t \in T$ と $p = [q_1, q_2, \ldots, q_n] \in P$ に対して $t \otimes p = (t \otimes q_1, t \otimes q_2, \ldots, t \otimes q_n)$ の評価期間 z は、$t \otimes q_1, t \otimes q_2, \ldots, t \otimes q_n$ の評価期間をそれぞれ $z_1, z_2, \ldots, z_n$ とすると $z = z_1 \vee z_2 \vee \ldots \vee z_n$ と表すことにします。z は $z_1, z_2, \ldots, z_n$ に分岐すると呼ぶことにします。$z_1, z_2, \ldots, z_n$ の時間的な順序は決まっていないものとします。

$t \in T$ の評価期間を z_0 とおきます。$Z_0 = \{z_0\}$ を t の評価空間と呼ぶことにします。$t \otimes p$ の評価期間が分割と分岐によって $z_0 = (z_{11} \gg z_{12} \gg \ldots \gg z_{1n}) \vee (z_{21} \gg z_{22} \gg \ldots \gg z_{2n}) \vee \ldots \vee (z_{m1} \gg z_{m2} \gg \ldots \gg z_{mn})$ と表されたとします。このとき $t \otimes p$ の評価空間は $Z_1 = \{z_{11}, z_{12}, \ldots, z_{1n}, z_{21}, z_{22}, \ldots, z_{2n}, \ldots, z_{m1}, z_{m2}, \ldots, z_{mn}\}$ とします。この1回の変換で得られる評価期間の集合 $\{z_{11}, z_{12}, \ldots, z_{1n}, z_{21}, z_{22}, \ldots, z_{2n}, \ldots, z_{m1}, z_{m2}, \ldots, z_{mn}\}$ を評価平面、z_0 をその頂点と呼ぶことにします。$t \otimes p^i$ の評価空間の各評価期間に対して、それを頂点とする評価平面に置き換えたものを $t \otimes p^{i+1}$ の評価空間とします。

$t \otimes p^n$ の評価空間 Z_n は $\leq$ によって順序集合になります。Z_n の各評価期間に対して、それを評価の時間の範囲とする項が対応しています。

ブレークポイント空間

C#のyield return文のようなものとして、項の後で出力変数を見ることができるようにすることができるものとします。このように設定された項に対応する評価期間をブレークポイントと呼ぶことにします。ブレークポイントは評価期間と評価期間の間と考えることもできます。ブレークポイントと評価期間の対を考えることによって、ブレークポイントと評価期間を同一視することもできます。

$t \otimes p^n$ のブレークポイント空間 B_n が評価空間と同様に定義できて、$\leq$ によって順序集合になります。ブレークポイントの分割、ブレークポイントの分岐、ブレークポイントの頂点とブレークポイント平面も同様に定義できます。ブレークポイント $b \in B_n$ に対して $B_n(b) = \{x \in B_n \mid x \leq b\}$ とおきます。これを b のブレークポイントパスと呼ぶことにします。ブレークポイントパスは $\leq$ によって全順序集合になっています。

$B_n(b) = \{b_1, b_2, \ldots, b_m\}, b_1 \leq b_2 \leq \ldots \leq b_m$ とし、$b_1, b_2, \ldots, b_m$ に対応する項をそれぞれ $t_1, t_2, \ldots, t_m$ とおきます。$t_1, t_2, \ldots, t_m$ の論理積を $\kappa_m = [t_1, t_2, \ldots, t_m]$ とおき、$b_1, b_2, \ldots, b_m$ にそれぞれ $\kappa_1, \kappa_2, \ldots, \kappa_m$ の結果（最後の項の出力変数の値）を対応させる写像を τ とおきます。写像 $\tau : B_n \to E$ によってリストのような構造を構築すると考えることができます。（C#でyield returnを含む関数がリストを構築すると考えることができますがそれと同様に考えます。図はイ

メージです。）

$$\begin{array}{ccccccc} b_1 & \xrightarrow{z_1} & b_2 & \cdots\cdots & b_{n\text{-}1} & \xrightarrow{z_{n\text{-}1}} & b_n \\ \downarrow\tau & & \downarrow\tau & \cdots\cdots & \downarrow\tau & & \downarrow\tau \\ \kappa_1 & \xrightarrow[t_1]{} & \kappa_2 & \cdots\cdots & \kappa_{n\text{-}1} & \xrightarrow[t_{n\text{-}1}]{} & \kappa_n \end{array}$$

ズームインパス

この構造の構築が有限の時間で終わるとすると、さらにこの構造の時間的な変化を考えることができます。これをこの構造に対する評価期間と考えることにより、複数の評価期間を持つ構造を考えることができます。（順序集合の直積として順序集合 $B_1 \times B_2 \times \ldots \times B_m$ とを考えます。図はイメージです。）

$$\begin{array}{ccc} B_1 \times B_2 \times \ldots \times B_m & \xrightarrow{b_1 \times b_2 \times \ldots \times b_m} & B_1 \times B_2 \times \ldots \times B_m \\ \downarrow{\scriptstyle \tau_1 \times \tau_2 \times \ldots \times \tau_m} & & \downarrow{\scriptstyle \tau_1 \times \tau_2 \times \ldots \times \tau_m} \\ T_1 \times T_2 \times \ldots \times T_m & \xrightarrow[t_1 \times t_2 \times \ldots \times t_m]{} & T_1 \times T_2 \times \ldots \times T_m \end{array}$$

あるブレークポイント b_1 からその頂点 b_2 、b_2 の頂点 b_3 、のように次々に頂点をたどっていくと、最終的に全体の頂点 b_m に到達します。このとき $b_m , b_{m\text{-}1} , \ldots , b_2 , b_1$ の列をブレークポイント b_0 のズームインパスと呼ぶことにします。このズームインパス $b_1 , b_2 , \ldots , b_m$ （添数を付け直します）は直積 $B_1 \times B_2 \times \ldots \times B_m$ の元と考えることもできます。

ある論理プログラム p による（0回、1回、… n 回の）有限回のズームインによって得られるズームインパスの全体を ZP_p とおきます。ズームインパスに式を対応させる写像を考えることにより、木のような構造を表すことができます。ズームインパスにはブレークポイントが対応しているので、ズームインパスの構造を無視してブレークポイントと考えることにより、リストのような構造を表すことができます。

H#/Lシステム

H/Lシステムのブレークポイントに対応する位置に、「 $set\,(o, x)$ 」という記述を書くことができるようにします。項 t_1 と t_2 の間にブレークポイントがあるとき

$\ldots, t_1, set\,(o, e)\,, t_2, \ldots$

と書くことができるようにします。o はある定数で、これを論理Fオブジェクトと呼ぶことにします。（これはオブジェクト指向的なオブジェクトを表すものと考えることができます。）e は式で、この位置のブレークポイントにこの式 e が対応することを表します。この記述によって論理Fオブジェクトを構築することができます。このような関数を論理Fコンストラクタと呼ぶことにします。（図はイメージです。）

$$\begin{array}{ccccccc} b_1 & \xrightarrow{e_1} & b_2 & \cdots\cdots & b_{n\text{-}1} & \xrightarrow{e_{n\text{-}1}} & b_n \\ \downarrow\tau & \downarrow set(o, e_1) & \downarrow\tau & \cdots\cdots & \downarrow\tau & \downarrow set(o, e_{n\text{-}1}) & \downarrow\tau \\ \kappa_1 & \xrightarrow[t_1]{} & \kappa_2 & \cdots\cdots & \kappa_{n\text{-}1} & \xrightarrow[t_{n\text{-}1}]{} & \kappa_n \end{array}$$

論理Fオブジェクトを使うために「 $get\,(o, e)$ 」という記述も書くことができるようにします。（図はイメージです。）

$$\begin{array}{ccccccc} b_1 & \xrightarrow{e_1} & b_2 & \cdots\cdots & b_{n\text{-}1} & \xrightarrow{e_{n\text{-}1}} & b_n \\ \downarrow\tau & \downarrow get(o, e_1) & \downarrow\tau & \cdots\cdots & \downarrow\tau & \downarrow get(o, e_{n\text{-}1}) & \downarrow\tau \\ \kappa_1 & \xrightarrow[t_1]{} & \kappa_2 & \cdots\cdots & \kappa_{n\text{-}1} & \xrightarrow[t_{n\text{-}1}]{} & \kappa_n \end{array}$$

論理Fコンストラクタ $f_1(o_1, e_1)$ で構築した論理Fオブジェクトを $f_2(o_1, e_2)$ で使うとき

$\ldots, f_1(o_1, e_1), set\,(o_2, e'), f_2(o_1, e_2), \ldots$

のように書くことができるものとします。この記述によって、論理Fオブジェクトの時間的な変化を表すことができます。順序集合の直積を考えることにより論理Fオブジェクトは1つにすることができます。

このようにH/Lシステムを拡張すると、リストのような構造を表すことができます。また、setがプログラム上の位置を記録できるもの、getがプログラム上の位置を参照できるものと考えることにより、木のような構造を表すことができます。このように拡張したものをH♯/Lシステムと呼ぶことにします。

結論

論理プログラミングにプログラム上の位置を記録する機能を追加することにより、ある再帰的な構造を表すことができます。

今後の課題

プログラム上の位置を記録する方法については、論理Fオブジェクトが単に定数と考える方法では実現できないかもしれないので、さらに詳しく考えていく必要があります。

オブジェクト指向言語の関数型言語的機能を実現するため、関数型言語でも同様の構造を表す方法についても考える必要があります。

参考文献

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/

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

履歴

2014年1月27日 Revision 1.00