数理解析研究所講究録
236巻 1975年 204-211

数式処理における数式の型について

津田塾大学　渡辺隼郎

§1. 動機と目的

数式処理プログラムを処理するシステムが会話型であるか否か，解釈型であるか否かは，数式処理とどのようにかかわっているだろうか。解釈型は解釈者以外はすべてデータと見なす，一方ALGOL-60のような非解釈型の言語では使用者が書いたものがプログラムで，これがデータを使い，実行時にプログラムとデータが入れ換わるということはない。

ところが数式処理においてはデータが数式であり，これを記述する言語が非解釈型の場合でも，プログラム中にも数式があるから，本来データとプログラムの区別はない。しかし実行の一段階毎に常に解釈を繰り返す必然性は，計算の途中経過が全く予想つかない場合を除いてない。会話型の場合，機械が出力する中間結果を見て，人間が入力を入れる場合，人間の熟考を要するものは一括処理の反復使用で間に合う。従って中間結果を再びプログラムに使うことに会話型の本質の一つがあると見なせる。この中間結果をプログラ

ムに使う時と場所，すなわち実行時の解釈と同等のものを要する時と所を最少限にすることはシステムの効率上重要であろう。　例えばパラメタを2つ持つ数式処理手続き $\Phi(\alpha, \beta)$ を考えよう。　$\Phi(A, B)$ は $\Phi(A, \beta)$ を実行した結果の $\varphi(\beta)$ の β に B を代入した結果とする。　一つの A_i に対して B_j が複数個でそれが大きいなら，$\Phi(A_i, B_j)$ の値を B_j の数だけ計算するには，$\varphi(\beta)$ を一度求めておいて，後はこれを利用するのが良いであろう。　これを実現するには $\varphi(\beta)$ を実行時にコンパイル（動的コンパイル）して，結果を実行すればよい。

あるいは同じことであるが $\varphi(\beta)$ だけを解釈する目的でプログラムを作ればよい。　それでは $\varphi(\beta)$ のような部分をどうして見つけるか。

動的コンパイルを行う場所の指定方法の一つはプログラム中に言語で $DC(f)$ のように f を動的コンパイルすべきてとを直接書くことであり，他の一つは我々の方法，つまり数式に型を導入して，特定の型を有する変数は常に動的コンパイルするという方法である。　数式の型は多項式，解析関数，あるいは x^2-x-1 といったどれをも型とするものであってほしい。　それは数式の型の一致，より具体的な型といった関係の真偽判定が意味論的にやさしく出来ることを目指すためである。　数式の型の間に我々は順序関係を導入し，て

れを，数式の演算子が働く働かないの判定条件に使う心づもりである．　例えば $f(x)$ を多項式とするとき，$\int f(x)dx$，$\int (x^2-1)dx$　の第一の式の $\int$ は働かないと第2の式の $\int$ は働くと考えるのである．

§2．数式と数式の型

数式処理用の言語として ALGOL-60 に次の機能を追加したものを考え，仮に Lβ と呼ぶことにする．　(1)型の定義　(2)演算子の定義　(3)数式に関する代入文　(4)型の宣言

Lβ で書かれたあるプログラムのあるブロック S を考え，S で有効な型の定義の集合を DT(S)，S で有効な演算子の定義の集合を DO(S) とする．　S における代入文は f:=g ここに f と g は数式，の形である．　数式は次の a～d のいずれかである．　(a)整数　(b)変数　(c)複合数式　(d)鎖　複合数式は (2·1) の形，鎖は (2·2) の形をしている．

(2·1)　$(O, f_1, \cdots, f_m)$　O：演算子　f_i：数式

(2·2)　$(\underline{chain}, f_1, \cdots, f_{mc})$　f_i：数式　mc：変化する．

但し鎖の場合 f_i はすべて型（後に述べる）は同じとする．

S の数式に出て来る変数は型の宣言を受けている必要があり，従って特定の型をその属性として有する．　S で記述可能な数式の集合を F(S) とする．　$F(S) \ni f$ のとき，f の型

T(f) を次のように定める。 f が整数のとき T(f) = f, f が変数のとき T(f) = 対応する型宣言子, f が複合数式 (2·1) のとき T(f) = (0, T(f₁), ---, T(fₘ)), f が鎖(2·2) のとき T(f) = (chain, T(f₁), ---, T(fₘc))。

Lβ に組込みの宣言子は integer と letter で, その他は

(2·3) define type ide as (0, tp1 id1, ---, tpn idn)

の形の型の定義により型宣言子 ide を定義する。 ここに, tpi は定義がどこかで与えられる型宣言子, idi は名前, ide はアンダーラインつきの名前である。 定義 (2·3) はさらに

(2·3′) ide → (0, tp1 id1, ---, tpn idn)

なる置換え規則を定める。 integer と letter は置換え規則

(2·3″) integer → <整数> (2·3‴) letter → <型宣言子>

を定め, tp を任意の型宣言子としたとき chain tp は

(2·3⁗) chain tp → (tp, ---, tp), tp の個数は任意,

を定める。 RR(S) = {rr; rr は δ∈DT(S) の定める置換え規則}, T(S) = {T(f); f∈F(S)} とおく。

T(S) の中に順序関係を次のように導入する。 t₁, t₂∈T(S) としたとき, t₁ = t₂ ⇔ t₁ と t₂ は文字列として同じ, t₁ < t₂ ⇔ t₁ の中の型宣言子に RR(S) を1回以上有限回繰り返して t₂ を得る。 どちらでもないとき t₁ と t₂ の間には関係が成り立たないとする。 さて T は F(S) を

T(S) の上へ写す写像であるが，一対一ではない。 T(f1) = T(f2) のとき f1 と f2 は同じ型の数式であるといい，T(f1) < T(f2) のとき f2 は f1 より具体的な数式という。

§3. 数式の評価と代入文

あるブロック S の代入文 f:=g ，(f, g は数式) を考えよう。 一般に数式は (1) プログラム数式と (2) 値数式，とに分れる。 (1) はプログラム数式の名の如く，プログラム中に書かれた数式の文字列そのものを指し，(2) は (1) が取る値としての数式を指す。 値数式は前値と後値とから成る。 プログラム数式 f の前値を PrV(f) で，後値を PsV(f) で表わす。 F(S) をそれ自身へ写す写像 eval を考え，PsV(f) = eval(PrV(f), f) と定義する。 PrV(f) は次のように定める。 f が整数のとき PrV(f) = f，f が (2·1) の形の複合数式のとき PrV(f) = (0, PrV(f1), --, PrV(fn)) ，f が鎖のとき PrV(f) = (chain, PrV(f1), ---, PrV(fnc)) ，f が型の宣言を受けてまだ一度も変数 f が左辺にある代入文の実行が行われていないとき PrV(f) = f， f を代入文 f:=g の実行後まだ他の f:=--- なる代入文の実行が行われていないので，f が変数であるとき PrV(f) = PsV(g)。

ここで変数につって説明しよう。 tR を型宣言子とするとき

tp idi なる型の宣言によつ idi なる名前で型 tp を持つ変数が定義されたことになる。 chain tp idi によつ，idi なる名前を持ち型 chain tp を有する変数と idi(k) なる名前を有する変数が定義されたことになる。 idi と idi(k) は関係 idi = (chain, idi(1), idi(2), ---, idi(k), --) で結ばれている。

さて PsV(f) はどのようにして評価するか。 まずこれがいわゆる数式の値の評価に相当するものであることに注意しよう。 f はプログラム数式とする。 f が整数のとき，PsV(f) = eval(PrV(f), f) = eval(f, f) = f, f が鎖のとき PsV(f) = (chain, PsV(f1), ---, PsV(fmc))。 f が複合数式である場合について説明しよう。 色々準備をする。 DT(S) に属する演算子 θ の定義

(3·1) define operation(θ, tp1 f1, ---, tpi fi, fi+1, ---, fm) as
tpr procedure theta <手続きの本体>

ここに tpr は型宣子 theta は名前でもちろん勝手なものでよい，のある複合数式 (θ, f1, ---, fm) を活動型数式，そうでない複合数式を休止型数式と呼ぶことにする。 f が複合数式または鎖のとき，fi を f の直接部分数式という。 i=0,1, ---, n-1 に対して $f^{(i+1)}$ が $f^{(i)}$ の部分数式であるとき，$f^{(m)}$ は $f^{(0)}$ の部分数式であるという。 その部分数式に活動型数

式を含まない数式を素な数式という。　またLβに組込まれた演算子の定義は整数の四則である。　ただし除算は商だけを答とする。

さてfが複合数式のときPsV(f)＝eval(PrV(f)，f)であるが，この右辺を説明しよう。　右辺はLβ-コンパイラがfをコンパイルして実行した結果である。　すなわち，まずfの部分数式で一番左にある素な活動型数式を取出す。　それを$(\theta_i, f_{i1}, \cdots, f_{ik})$とする。　θ_iに対する定義がDT(S)の中にあるから，対応する手続きがある，その名前をthetaiとする。　Lβ-コンパイラは，入力パラメタ PrV$(\theta_i, f_{i1}, \cdots, f_{ik})$をもってthetaiを呼び出し，その結果をLβ-コンパイラが生成した変数v(i)，その型はthetaiの型である，に代入する目的プログラムを生成する。　さらにfの$(\theta_i, f_{i1}, \cdots, f_{ik})$をv(i)でおきかえる。　そしてこの過程を新しいfについて繰り返してf中にもはや活動型の数式がなくなるまで続ける。　さてこのようにして生成された目的プログラムを実行した結果のfをPsV(f)と定義する。　それではLβ-コンパイラが生成した手続き呼び出しthetai(PrV$(\theta, f_1, \cdots, f_m)$)は次のように動くことを説明しよう。　対応するθの定義は(3·1)とする。　$(\theta, t_{p1}, \cdots, t_{pi}, f_{i+1}, \cdots, f_m) \leq T(PrV(\theta, f_1, \cdots, f_m))$が真であるならthetaiへの呼び出しが実

行され，偽であるなら thetai(PrV(0, f1, ---, fm)) = PrV(0, f1, —, fm) とする。

f が変数の場合，T(f) ≠ letter なら PsV(f) = f とする。 T(f) = letter なら PsV(f) = PsV(PrV(f)) とする。右辺の PrV(f) は実行時でないと定まらず，PsV はコンパイルを意味するから，letter を型とする変数を含む数式が動的コンパイルの対象となる。

最後に代入文の意味について説明しよう。 f:=g は T(f) ≦ T(PsV(g)) のときだけ実行可能であって，その意味は PrV(f) = (0, v1, --, vk, fk+1, ---, fm)，ここに vi は変数，fj は変数以外の数式，PsV(g) = (0, g1, --, gk, fk+1, --, fm)，ここに fi, gi は数式，とするとき，vi := gi , i=1, 2, ---, k の意味になる。

T(f) = chain tip のとき PsV(g) = (chain, h(1), ---, h(nc)) ならば f:=g は f(i) := h(i) i=1, ---, nc に等しい。 このとき，f の前値は次の代入文まで (chain, f(1), f(2), ---, f(nc)) となる。

以上.