データ型の同値性に関する一考察

金藤栄孝・山野紘一
（(株)日立製作所　システム開発研究所）

1．はじめに

Russellは、Cantorの古典的な集合論に基づくFregeの論理体系に於いて、「自分自身を要素として含まぬ集合の集合」といった無秩序な集合概念の使用に起因するパラドックスを回避する目的で、「型の理論(type theory)[1]」を提案した。プログラミング言語に於いても、Algol68やそれ以後の会話用を除く、殆ど総ての言語で、記述されたプログラムの可読性や信頼性の向上を主目的として、データ型の概念が導入され、それを言語で支援する為の機能（新しいデータ型を定義する為の構文etc.）が整備されて来た。プログラミング言語でも、データ型の概念が不完全であったAlgol60[2]やオリジナルPascal[3]では、Russellのパラドックスに相当するプログラムが構成できてしまう[4]。例えば、オリジナルPascalでは、

```
function  F(function  G): Integer;
begin
  if  G(G) = 1  then  F := 0  else  F := 1
end
```

となり、この函数Fの呼出し、F(F)、は、自己適用による再帰呼出しを生じ、停止しない。この問題は、函数仮パラメタの仮パラメタのデータ型を指定する言語では生じない（上の様なプログラムを記述できなくなる）。これは、型無しλ-算法が停止性を保証しないのに対し、一階の型付きλ-算法が停止性を保証する事に対応する[5]。

以上の様に、プログラミング言語に於いても、データ型の概念は、自己適用によるエラーの発生を防ぐ上で、Russellの型と同様の役割を果しており、プログラムの信頼性の向上を考える上で、重要となっているが、一般に、プログラミング言語にデータ型（以下、単に型と呼ぶ）を導入する際、先ず大切な事は、

(1)　型の同値性 ---- 2つの型は、どの様な場合に等しいと定めるか

という問題の扱いであり、さらに、これに基づいて、

(2)　型の適合性 ---- 式の表わす値は、どの様な場合に、その式が出現した文脈で要求されている型を満足するとするのか

をどう考えるのかという2点である。これらの解によって、型を導入しようとしている言語の構文にまで、影響が及び得る。

そこで本報告では、2章で型の同値性の問題に対する既存のアプローチ（解決案）を示し、3章では、それらのアプローチの問題点について検討する。次に4章で、新しい型の体系の構成を示し、5章でこの型の体系の上での型の同値性の解決案を提案する。6章ではこの同値性の基準に基づく型の適合性の基準を提案し、7章で、これらの型の同値性／適合性の基準に従った言語に於ける型の運用法について考察する。

2．型の同値性に対する既存のアプローチ

型の同値性の基準としては、これまで、大別して、以下に示す2通りの考え方があった。

2.1 構造同値

型の名前は、その型の定義式の略記法に過ぎず、型の定義式が同一の構成なら、同一の型を表わすと考える。

この基準を採用した例としては、Algol68[7], Euclid[8], Clu[9], ML[10] 等が挙げられる。

2.2 出現同値（しばしば、名前同値と呼ばれる）

型の定義式が同じ構成をしていても、相異なって出現した定義式は、相異なる型を表わすと考える。

この基準を採用した例としては、Ada[11], ISO規格案のPascal[12]等を挙げる事ができる。

なお、Tennent[4]は、この基準が、しばしば名前同値と呼ばれる事は不正確であり、出現同値と呼ぶ事を提案している。本報告では、出現同値の方を用いる。

2.3 両基準の比較例

以上の2つの基準の違いは、例えば、次の様な例に現われる（ここで、int は、整数に対する標準の型名とする）。即ち、

```
type Count = int;
type Price = int;
```

という2つの型の定義がある時、

(1) 構造同値 ···· Count, Price, int は、同一の型を表わす3つの別名と解釈する。

(2) 出現同値 ···· Count, Price, int は、各々、相異なる型を表わす名前と解釈する。

という違いがある。

3. 既存のアプローチの問題点

3.1 構造同値

2.3での例で示されている様に、同一の構成を有する型を区別する事ができない。例えば、Count型の値をPrice型の値として誤って用いる類のミスをエラーとして検出する事を不可能にする。

3.2 出現同値

出現同値の抱える問題は、大きく分けて2つある。それは、

- 多重定義の発生
- 無名型の取り扱い

であり、以下、各々について検討する。

(1) 多重定義の発生

2.3の例について言えば、「7」といった基本的な表記、「+」等の演算子名、等は、Count, Price, int いずれの型にも属し、必然的に多重定義（overloading）が生ずる。この多義性を文脈に基づいて解決できる様にする為に、複雑、かつ、ad hocな多数の規則と構文を文法書に追加せねばならぬ事は、Adaで知られている[11]。誤解のない様に言えば、構造同値を採用した言語で

も多重定義は起こり得る（ex. Algol68[7]）が、構造同値では、型が異なるのはデータ構造が異なる場合であり、出現同値で起こる様なデータ構造が同じで型名のみ異なる型の間での多義性の解決の様な微妙な問題はない。

出現同値では、データ構造が異なるという大きな差も、単に型名が違うだけの差も「型が違う」という同一の次元で扱う為に、多重定義を、関係するデータ構造の違いに基づいて整理し、その後に、型名のみの違いを考えるといった管理ができない点に問題がある。

(2) 無名型の取り扱い

出現同値では、異なって出現した無名の型（即ち、型定義式そのもの）は、総て相異なると考える。このために、例えば作用的言語の様な、函数も一階のデータとして扱う言語では、困った事になる。つまり、以下の様な函数の定義は、函数構造を持つ無名型の値を表わす定数名の定義となる。即ち、

```
function   F(N: int): int  =  N*N
```

は、

```
const      F: func(int, int) =  λN. N*N
```

[func：函数構造用の型構成子]

として扱われる。この時、問題となるのは、同じ函数を2回作用させる函数Twice、を、

```
function   Twice(G: func(int, int)): func(int, int)
             = λM. G(G M)
```

と定義した時、(Twice F)5、というTwiceの呼出しは、出現同値では、エラーとなる。何故なら、Twiceの仮パラメタの型も実パラメタFの型も、同一の構成、func(int, int)、を持つ無名型であるが、相異なる出現の為に、相異なる型となり、仮／実パラメタ間の型の不一致が生ずるからである。

しかし、この「func(int, int)」の様な無名型を、総て、独立した1個の型として、型名を付けて前もって型定義してから使用せねばならないとすると、実用上、極めて不便になる。例えば、上のFとTwiceの定義は、

```
type    IntFunc      =  func(int, int);
        IntFuncFunc  =  func(IntFunc, IntFunc);
const   F:  IntFunc  =  λN. N*N;
        Twice: IntFuncFunc  =  λG. λM. G(G M)
```

とせねばならず、函数は総て定数名として定義する必要がある(syntax sugarとしての函数定義文が使用できない)。

この例で示される様に、出現同値では、無名の型は殆ど有効に使用できない。しかし、総ての型に名前を付けて用いるのは、書き易さを損うばかりか、読み易さまでをも損なう事になり、型を用いる目的からすれば、本末転倒の感を免れない。

出現同値に於ける、この様な無名型の問題の基本的原因は、出現同値では、型定義式に参照透明性がない事である。

3.3 問題点の本質

構造同値では、折角、プログラマが区別しようと別の型名を付けた努力を無視する点に問題がある。出現同値では、プログラマが独立した1個の型と考えていないものまで勝手に一意な型名を与えて区別してしまう御節介さが問題である。

結局、いずれも、プログラマの意図に反する点が問題なのであり、この点を鑑

み、本報告では、同値性の新しい基準として、意図同値（intentional equivalence）を提案する。

4. 型の体系の構成

ここでは、意図同値を組み込む為の型の集まり（型の体系と呼ぶ）の構成モデルを考える。その準備として、この構成を考える上で、適用対象のプログラミング言語が満たしていると仮定している条件について述べる。

4.1 適用対象言語に対する条件

(1) 適当な基本型（例えば、int, real, bool等）を持つ。

(2) 適当な型構成子（例えば、prod(直積)、sum(直和)、func(函数)等）を持ち、各型構成子の仮型パラメタ数は固定とする（上の3つはいずれも2個とする）。

(3) 型の定義は、1段階毎に行なう。従って、例えば、

```
type T = prod( func(T11, T12),
               func(T21, T22))
```

の様に2段階を一気に定義する事は許さない（この制約は、議論の単純化の為で、後で取り除く）。

(4) 直和構造の型の値を、その成分型の値からinjectして作る時は、どの成分型（ここでは、sumは2個の仮型パラメタを持つとしているので、例えば、オ1の成分型かオ2のか）からのinjectionかを明示する。

(5) 列挙型の定義は含んでも良い。但し、列挙定数名については、多義性を生じない為に、使用時に属する型を指定する目的で、型指定子（Adaの型限定式での「型名'」相当）を必ず前置する等の規制を加える。

(6) 既に定義された型と同じ内容の別の型を定義する事を許す。これは、Adaの派生型定義と同様のものである。

(7) 部分範囲は、独立した型とは認めない。

以上の制限条件は、特異なものではない。(1)、(2)、(4)については、ほぼML[10]での型の取り扱いから、型多形性を除いたものと同等である。(5)の列挙定数名の扱いに関しては、整定数等の基本的な表記と同等に扱おうとするAdaの方針では様々な困難が生じ、本来、同一レベルで扱う事自身が不当であると指摘されている[13)~15)]。(6)については、既に3.1節で述べた理由による（派生型は廃止すべきだという見解[16)]もあるが、ここでは含める）。(7)は、Adaの型と副型の区別と同じ理由で、静的に値のmembershipの判定が可能な集合のみを独立した型として扱う事にする。

4.2 型の体系

今、仮に、以下の説明で用いる型定義文の構文は、下記の様なものであるとする（記法はAN記法[17)]による）。

```
<型定義文> ::= type ( <型名> = <型定義式> ) {,}...
<型定義式> ::= <型構成子呼出し式> | <列挙型定義式>
             | <派生型定義式>
<型構成子呼出し式> ::= <型構成子名> ( <型名> {,}... )
```

<列挙型定義式> ::= enum (<列挙定数名> {,}...)
<派生型定義式> ::= new <型名>

さて、この時、

(1) 基本型の各々について、対応する汎型(名前は各基本型名の前に「U」…この文字は、今、考えている言語の文法で現われない文字とする…を付けたものとする)を考え、各基本型は、対応する汎型に「直属する」と定義し、この関係を、「$\sqsubset_1$」で表わす。例えば、

real $\sqsubset_1$ Ureal　　あるいは、　int $\sqsubset_1$ Uint

等である。

(2) 以下の議論の為に、補助的な記法を幾つか定義しておく。
P, Q, R を型もしくは汎型(双方を表わす時は、以下、"型"と引用符で括る)とする時、

- $P = Q \triangleq$ PとQの名前が等しい
- $P \neq Q \triangleq \sim(P = Q)$
- $P \sqsubset Q \triangleq (P \sqsubset_1 Q) \vee (\exists R.((P \sqsubset_1 R) \wedge (R \sqsubset Q)))$
- $P \sqsubseteq Q \triangleq (P \sqsubset Q) \vee (P = Q)$

と定義する。なお、「$\sqsubseteq$」は、「属する」、「$\sqsubset$」は、「真に属する」と読む。もちろん、

$(P \sqsubset_1 Q) \Rightarrow (P \sqsubset Q) \Rightarrow (P \sqsubseteq Q)$

である。

(3) 型定義で定義される型は、右辺の型定義式の表わす汎型に直属すると定義する。ここで、

(a) 型定義式が、型構成子呼出し式になっている時は、その構成に 1:1 に対応し、名前がその型定義式に「U」を前置した汎型を表わすと定義する。例えば、prod(real, real)、は、汎型、Uprod(real, real)、を表わす。

(b) 型定義式が、列挙型定義式の時は、先ず、「U列挙型名」という名前の汎型が定義され、この汎型に直属する型として、左辺の型が定義されると定める。

(c) 型定義式が、派生型定義式の時は、new の後の型が直属する汎型を表わすと定義する。

であり、例えば、

```
type   Plane_coord = prod(real, real),
       Colours     = enum(RED, YELLOW, BLUE),
       Price       = new int,
       Count       = new int
```

という型定義では、

Plane_coord $\sqsubset_1$ Uprod(real, real)
Colours $\sqsubset_1$ UColours
Price および Count $\sqsubset_1$ Uint

が成立する。

(4) 任意の"型"、P, Q, R、について、以下が成立する。

- $(P \sqsubset Q) \wedge (Q \sqsubset R) \Rightarrow (P \sqsubset R)$
- $P \sqsubseteq P$

(5) 型構成子呼出し式、$\mathcal{E}_X$、の表わす汎型を $\mathcal{U}_X$ とする。$\mathcal{E}_X$ に現われる任意の"型"、X、をその直属する"型"、Y(即ち、$X \sqsubset_1 Y$)、で、ちょうど1回だけ置換した式、$\mathcal{E}_Y$、に「$\mathcal{U}$」を前置してできた汎型を $\mathcal{U}_Y$ とすると、先の「$\sqsubset_1$」について、

$$\mathcal{U}_X \sqsubset_1 \mathcal{U}_Y$$

が成立すると定義する。例えば、

$$\underline{\mathcal{U}prod}(int, int) \sqsubset_1 \left\{ \begin{array}{l} \underline{\mathcal{U}prod}(\mathcal{U}int, int) \\ \underline{\mathcal{U}prod}(int, \mathcal{U}int) \end{array} \right\} \sqsubset_1 \underline{\mathcal{U}prod}(\mathcal{U}int, \mathcal{U}int)$$

である。

(6) 次章以後の議論の為に、この"型"の体系の上での、上/下限を定義しておく。P, Q, R, S は、"型"を表わすとし、

- PとQの上限、$P \sqcup Q$、は、
  $(P \sqsubseteq R) \wedge (Q \sqsubseteq R) \wedge \sim(\exists S.((S \sqsubset R) \wedge (P \sqsubseteq S) \wedge (Q \sqsubseteq S)))$
  なるRが存在すれば、そのR、存在しなければ、上限は存在しない
- PとQの下限、$P \sqcap Q$、は、
  $(R \sqsubseteq P) \wedge (R \sqsubseteq Q) \wedge \sim(\exists S.((R \sqsubset S) \wedge (S \sqsubseteq P) \wedge (S \sqsubseteq Q)))$
  なるRが存在すれば、そのR、存在しなければ、下限は存在しない

と定義する。

(7) 自分自身以外のいずれの"型"にも属さない"型"を「完全汎型」と呼ぶ。例えば、次節の例1の図1の"型"の体系では、

$\mathcal{U}int$ と $\underline{\mathcal{U}prod}(\mathcal{U}int, \mathcal{U}int)$ と $\underline{\mathcal{U}prod}(\underline{\mathcal{U}prod}(\mathcal{U}int, \mathcal{U}int), \mathcal{U}int)$

が該当する。

(8) 上述の関係、「$\sqsubseteq$」は、半順序関係である。半順序関係の満たすべき性質の内、反射律と推移律は、(4)より明らかであり、残る反対称律、

$(X \sqsubseteq Y) \wedge (Y \sqsubseteq X) \Rightarrow (X = Y)$ についても、

$(X \sqsubseteq Y) \Rightarrow$ (Xの名前中の「$\mathcal{U}$」の個数 $\leq$ Yの名前中の「$\mathcal{U}$」の個数)

に注目すれば、明らかである。

4.3 例

以下の型定義に対する"型"の体系を図1に示す。なお、図中、直線で継がれているのが「$\sqsubset_1$」の成り立つ組合せで、図で下の"型"が上の"型"に直属する。

```
type   A   = new int,
       B   = new int,
       AA  = prod(A, A),
       BB  = prod(B, B),
       AB1 = prod(A, B),
       AB2 = prod(A, B),
       AAA = prod(AA, A)
```

[図中、
「2」: 「$\mathcal{U}int$」の略
「π」: 「$\mathcal{U}prod$」の略]

4.4 "型"の体系(続)

ここでは、4.1での制限事項の緩和について検討する。

(1) 型構成子の仮型パラメタ数の可変化

現実のプログラミング言語の多くでは、直積や直和の成分型の数は任意である。これについては、各パラメタ数毎に型構成子が用意されていて、それ

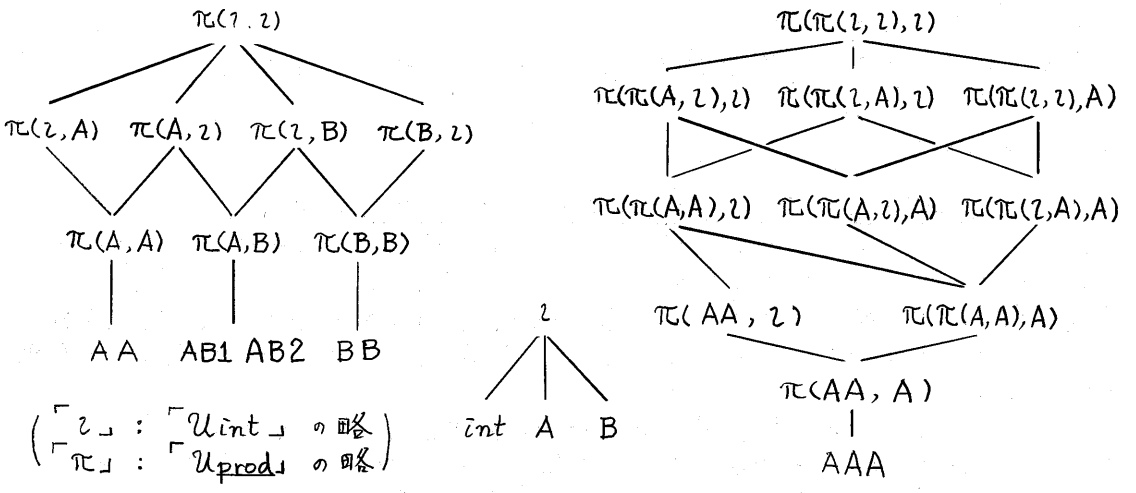

図1 "型"の体系の例

らをprodやsumという名前で見かけ上多重定義されて用いていると考えれば良い。即ち、パラメタ数をmで表わすと、例えば、prodNm やsumNmという型構成子が実際に使用されると考えれば良い。例えば、

prod(real, real, real) の表わす汎型は、
Uprod N 3 (real, real, real) となる。

[Nは U同様他で出現しない「文字」。]

(2) 直積/直和構造の型定義へのフィールド名の導入

フィールド名を扱うには、空のフィールド名(φで表わす)を考え、前述の関係、「⊏₁」、を、フィールド名の有無についても定義する。例えば、

prod(Re: real, Im: real) (Re, Imがフィールド名)

では、 Uprod(Re: real, Im: real) ⊏₁ { Uprod(φ: real, Im: real)
Uprod(Re: real, φ: real)

等々。

(3) 多段階まとめての型定義(4.1(3)の廃止)

例えば、次の型定義について考えると、

type AAAU = prod(prod(A,A),A) では、
AAAU ⊏₁ Uprod(Uprod(A,A),A) と考える。即ち、

型構成子呼出しでの引数として、型名でなく、型定義式が直接現われた時は、引数の型定義式は、対応する汎型を表わすと定義する。但し、列挙型定義式は、型構成子の引数として直接用いられない。これは、制限事項(5)に反する(列挙定数名を使う時に前置すべき型指定子の型名がない)からである。

5. "型"の同値性

以上の"型"に対する同値性の基準は、"型"の名前の等否に基づくと定義する。

この場合、型定義で名前を付けて定義した型は、出現同値でと同様、独立した型として区別され、構造同値の様に、プログラマの努力を無視する事はない。

逆に、出現同値で問題となる無名型については、同一の構成の型定義式は、何度出現しても、同一の汎型を表示する事により、型定義式は参照透明性を保持し

ており、トラブルは生じない。

この同値性の基準を、意図同値（intentional equivalence）と呼ぶ。

6. "型"の適合性

2つの"型"、X, Y、について、

"型"Xは、"型"Yに「適合可能」 $\Leftrightarrow$ $Y \sqsubseteq X$

と定義する。そして"型"Xが"型"Yに適合可能な時、かつその時のみ、"型"Xの値を"型"Yの値として転用できるとする。

この"型"の適合性を考慮すれば、完全汎型は、構造同値での型と同様に振舞う。つまり、ある完全汎型の値は、その完全汎型に属する任意の"型"の値として転用可能である。

ここで、本提案での"型"の扱いの直観的／実際的な解釈を示しておく。

(1) 型定義式の表わす内容、即ち、汎型の表わす内容

型定義式は、定義しようとしている集合の要素である値が持つべきデータ構造と、そのデータ構造に見合った基本演算の集まりとを表わす。

(2) 型名を付けて型定義をする事の目的

型定義式に名前を付けて、1つの独立した型として定義するのは、そのデータ構造の値を、特定の用途に限定させる為に行なう。例えば、

type Complex = prod(real, real)

という定義は、2つの実数より成る対を、特に複素数という特定の用途に限定して扱う為である。

(3) 半順序関係、「$\sqsubseteq$」、の表わす内容

"型"XとYについて、$X \sqsubset Y$、が成立するのは、

- X, Yは構造同値的に見れば、相等しく、
- Xは Yを構成している汎型…データ構造のみを表わす…を、型…固有の限定用途が対応している…に置き換えたもの

である。従って、Xは Yよりも、より明確に用途が限定されていると考えられる。

(4) 意図同値の目的

プログラマが、区別したいと意図する水準（データ構造のみか、特定された用途も含めてか）を素直に同値性に反映する事。

(5) "型"の適合性の意味

$X \sqsubseteq Y$、の時、"型"Yの値を"型"Xの値として扱えるというのは、まだ用途指定が不完全な値を、それに対し既に指定している用途と矛盾しない範囲で、必要に応じて用途を追加制限する事は許されるという事である。即ち、Yの方が、Xよりも、使用の自由度が大きいのである。逆にXの値をYの値として扱えない（$X \sqsubset Y$の時）のは、既にXの値に対し限定された用途は、明示的指定がない限り、自動的に無視する事は許さないという事である。

7. "型"の運用法

以上の"型"の同値性／適合性に基づく"型"の使い方について提案する。

(1) 多義性の扱い

(a) 基本表記

例えば、整定数は U_{int} の様に、基本表記は、適切な完全汎型の値とする。

(b) 列挙定数名

既に 4.1 (5) で述べた様に、列挙定数名は型指定子で多義性を解消して用いねばならない。

(c) 函数・演算子

型多形性 (type polymorphism)[6] の記述法を拡張し、以下の様な宣言を用い、多重定義でのパラメタ/結果の型についての構造的条件や相互間の束縛条件を表現できる (seq は、列構造の為の型構成子とする)。

overload

any E, S such that S ⊑ $\overbrace{\cup\text{-seq}(E)}^{(\cup_{\text{の concrete な表現}})}$ in function Cons(E,S): S

この例では、函数 Cons は、常に2つのパラメタを持ち、才2パラメタの"型" (S) の構造は才1パラメタの"型" (E) の列となっていなければならず (S ⊑ ∪-seq(E))、結果は才2パラメタと同じ"型"である事を指示している。この様に、適切な多重定義の範囲を指定する事ができる。

(2) 代入文

V := E、という代入文は、ここでの"型"の適合性に基づけば、

Vの"型" ⊑ Eの"型"

の時、正しいと考える。

(3) 式の"型"の決定法 (以下、e_1, … は式とし、その"型"は T_1, … とする。)

(a) 集成式

(i) 直積データ … 成分値の式を「[」と「]」で括るとする

直積データの集成式、[e_1, e_2]、の値の"型"は、$U_{prod}(T_1, T_2)$ である。

(ii) 列データ … 各要素値の式を「<」と「>」で括るとする

列データの集成式、<e_1, e_2, …, e_n>、の値の"型"は、

- n = 0 の時: 任意の"型" T について $U_{seq}(T)$ と適合できる"型"
- n = 1 の時: $U_{seq}(T_1)$ 汎型
- n > 2 の時: $T_1 \sim T_n$ の下限、T、が存在すれば、$U_{seq}(T)$ 汎型。なければエラー。ここで、T は次の通り。

  T ← T_1;

  for i from 2 to n step 1 do T ← T ⊓ T_i

である。

(b) 多重定義された函数・演算子等の呼出し

実パラメタが、(1)-(c) の型多形的定義での仮パラメタの構造的条件 (列構造か? 等) を満たすか検査し、正しければ、相互の束縛条件で、同一の"型"のはずのものについて、下限が存在するか否か判定する。例えば、

Cons(e_1, e_2) では、

- $T_2 \sqsubseteq U_{seq}(T)$ なる T (最も下限のもの) に対し、
- $E_0 \triangleq T_1 \sqcap T$ なる E_0 と
- $S_0 \triangleq U_{seq}(E) \sqcap T_2$ なる S_0 を求め、

Cons : $E_0 \times S_0 \to S_0$ と決定される。

(c) 再帰的な型との適合性

本報告で述べた"型"の体系は、再帰的に定義された型に対しては、無限

になる。従って、式の"型"が再帰的な型に適合するか否かの判定は、

- 式の"型"、T、を(a),(b)等を用い、ボトムアップに決定し、
- Tの名前の長さ、ℓ_T、を求め、
- 文脈の再帰的な型の体系上を"型"の名前の長さの短い方から、ℓ_Tのものまで順に調べ、Tがあれば、OK、なければ、エラー、となる。

ここで、"型"の名前の長さは、例えば、

「υ」の数 + 型名の数 + 型構成子名の数 + 「(」の数 + 「)」の数

と定義すれば良く、任意の"型"、A, B、について、

$$A \sqsubset B \Rightarrow (A\text{の名前の長さ}) < (B\text{の名前の長さ})$$

が成立している。

8. おわりに

新しい、データ型の同値性の基準、意図同値、を提案し、そのもとでのデータ型の取り扱いについて検討した。この結果、出現同値の肌面細かい型の使い分けを許しつつ、構造同値的な概念としての汎型の考え方を導入し、無名型や多義性の問題を、合理的に扱える事を示した。なお、我々とは別のアプローチであるが型定義時に、出現同値/構造同値いずれで扱う型かを指定し、両同値性基準の混用を図った例がある[18]。

今後、適合性判定の為の効率の良いアルゴリズムの考察等、インプリメンテーションについて検討を進めたい。

[参考文献]

1) Hatcher, W.S. "The Logical Foundation of Mathematics", Pergamon, Oxford (1982).
2) Naur, P. "Revised Report on the Algorithmic Language ALGOL60", Commun.ACM, 6, 1-20 (1963).
3) Jensen, K. et al. "Pascal User Manual and Report"(2nd ed.), Springer-Verlag, New York (1974).
4) Tennent, R.D. "Principles of Programming Languages", Prentice-Hall Intern'l, London (1981).
5) Fortune, S. et al. "The Expressiveness of Simple and Second-Order Type Structures", J.ACM, 30, 151-85 (1983).
6) Milner, R. "A Theory of Type Polymorphism in Programming", J.Comput.Sys.Sci., 17, 348-75 (1978).
7) van Wijngaarden, A. et al. "Revised Report on the Algorithmic Language ALGOL68", Springer-Verlag, Berlin (1976).
8) Lampson, B.W. et al. "Report on the Programming Language Euclid", SIGPLAN Notices, 12(2) (1978).
9) Liskov, B. et al. "Clu Reference Manual", Lecture Notes in Comput.Sci., 114 (1981).
10) Gordon, M.J. et al. "Edinburgh LCF", Lecture Notes in Comput.Sci., 78 (1979).
11) US-DoD "MILITARY STANDARD Ada Programming Language", ANSI/MIL-1815A (1983).
12) Addyman, A.M. "A Draft Proposal for Pascal", SIGPLAN Notices, 15(4), 1-66 (1980).
13) Moffat, D.V. "Enumeration in Pascal, Ada, and Beyond", SIGPLAN Notices, 16(2), 77-82 (1981).
14) Welsh, J. et al. "Ambiguities and Insecurities in Pascal", Software Practice & Experience, 7, 685-96(1977)
15) Tennent, R.D. "Another Look at Type Compatibility in Pascal", ibid, 8, 429-37 (1978).
16) Hilfinger, P.N. "Abstraction Mechanism and Language Design", MIT Press, Mass. (1983).
17) 島内「プログラム言語論」, 共立出版, 東京 (1972).
18) Van Deusen, M. "Types in Red", SIGPLAN Notices, 16(12), 27-38 (1981).