Gradia: 式レベル型注釈による段階的型付き Lisp 系言語

梶塚 太智[a)]

概要：本稿では, 式レベルで型注釈を可能にする段階的型付けの Lisp 系言語「Gradia」を提案する. 従来のプログラミング言語では, 型注釈は変数や関数にのみ付与されることが一般的であり, 式全体に対して細かく指定することは難しい. Gradia は, プログラムを構成する S 式に対してリストやアトムごとに型注釈を付与できる点が特徴であり, 式レベルの型指定をサポートすることで, より柔軟かつ安全なプログラム設計を可能にする. 本稿では, Gradia の型システムの設計, 実行時の挙動, および式レベルでの型注釈による利点を考察し, プログラムの静的保証と柔軟性を両立させるための新しいアプローチを示す.

1. はじめに

現代のソフトウェア需要は型安全性にあると言えよう. 例えば Rust や TypeScript など, 最近の言語はパフォーマンスと静的型付けを前面に押し出している.

動的型付け言語である Lisp では, declare 関数を使って型注釈を付与することが可能だが, 変数や関数にしか付与することが出来ない. Lisp はその性質上, ネストが大きくなりやすく, S 式のアトムやリストそれぞれに型注釈を付与することが, 現代の型安全性を求めるソフトウェア需要にとって必須であると考え, この研究に至ったのである.

2. 型システム

Gradia の型システムは, Gradual Typing, いわゆる段階的型付けを採用している. 型注釈を省略すると, 動的に式の結果の型が決まり, 従来の Lisp のように使用できる. いっぽう型注釈を付けると, 実行前に式の結果の型が決まり, 厳密に型チェックをして, 式の結果の型が期待されたものと違えばエラーを吐く.

これにより, 動的型の柔軟性を保ちながらも, 必要に応じて型注釈を付与し型安全性を高めることが出来る.

3. 挙動

以下のコードは, 単純に受けとった値を 2 倍する x2 関数を Gradia で実装したものである.

```
(define '(x2 n) '(* n 2))
```

型注釈が無いため動的型になり, 数値はもちろんのこと, 文字列を与えて呼び出しても暗黙の型変換されて実行される.

以下の出力例は REPL で実行した物である.

```
> (x2 3)
6
> (x2 " 4")
8
```

x2 関数の定義で, 引数に型注釈を付けると以下のようなコードになる. この定義においては, 引数の n が数値型であることが期待される.

```
(define '(x2 n:number) '(* n 2))
```

型注釈があるので厳密に型チェックされ, 文字列を与えると期待された型と違うので以下のようにエラーを返す.

```
> (x2 5)
10
> (x2 " 6")
Error! the passed argument value `" 6"`
    is different to expected type `
   number` of the function
```

4. 文法

Gradia の文法を, EBNF で表すと以下のようになる.

```
<sexpr> ::=  <atom> | <list> [":" <type
   >]
<type> ::= "number" | "string" | "bool"
    | "symbol" | "function"
```

式と型注釈の区切りにコロンを使う方法は, Python や Rust など多くの言語で見られ, プログラマにとって非常に直感

a) kajizukataichi@outlook.jp

歴であると言えよう.

先ほどの x2 関数で, 全ての式に型注釈を付けると以下のようになる.

```
(define:function '(x2:symbol n:number):
    list '(*:function n:number 2:number)
    :list):function
```

Gradia の式レベル型注釈は, S 式のリストだけではなく, 変数やリテラルまでの詳細に型注釈を付与することが出来るのである.

5. 式レベル型注釈の是非

式レベルで型注釈を付けられることには多くのメリットがある.

一般的な言語では, 型注釈は変数に付けるものだが, 一つの変数に纏めようと長い式でメソッドチェーンや関数のネストの入れ子が膨大になり, 特に動的型付け言語の場合はバグの温床になる.

例を挙げよう. 以下は Python のコードである.

```
numbers: list[int] = map(lambda n: n *
    2, [1, 2, 3])
```

側から見ると, 型注釈から整数型のリストが返ると思われる. しかし結果は Map オブジェクトになる. Python では型注釈をしても型チェックがされるわけではないのである.

正しく処理するのは, list 関数で型変換をする必要がある.

```
numbers: list[int] = list(map(lambda n:
     n * 2, [1, 2, 3]):map)
```

しかし, Map オブジェクトが返ることを, Python は式レベル型注釈で明示できないため, 何の為に list 関数で囲っているのかがよく分かりにくいのである.

そのような問題を解決する為にも, 式レベル型注釈のアイデアは, Gradia だけに留まらず, 様々な言語にとって有用な機能になることが期待できる.