プログラムの再帰的意味論

```
Def prime():
    for n in range(2, 10000): isprime=True
        for i in range(2, n):
            if n % i == 0:  isprime = False
        if isprime: print n
```

ただの文字列に見えるが、コンピュータに慣れた人が動作をトレースすると「10000以下の素数を小さい順に表示する」という“意味”が判明する。

プログラム文字列の実行結果やその過程について、数学的構造で表すことを「プログラム意味論」と呼ぶ。実行前に、その結果の収まる範囲を予測したり、与えられた2つのプログラムが本質的に同じものかを判定するなど、プログラムの性質や効率性の検証を目的としている。“意味論”と呼ばれる理由は、構造や数理モデルへ対応付けて、意味の理解を図るからだ。

プログラム自体は、おおざっぱに言えば「ある領域からある領域への写像」と言える。冒頭の例は、自然数全体から素数全体への写像である。しかしながら意味付け方法は無数に存在し、極端な例では、紙上のドット集合から、コンピュータ内の電圧変化への写像とも言えてしまう。有益な情報を取り出すには、プログラムを“美しい数理モデル”で捉える必要がある。

プログラムの中で、自分自身の呼び出し構造を持つものを再帰プログラムと呼ぶ。プログラマーにとっても恐ろしいことだが、再帰プログラムは一度実行すると停止しない可能性がある。

```
Factor(n):=if n=0 then 1 else n×Factor(n-1)
```

これは「自然数の階乗を計算する」再帰プログラムの例だが、「ある汎関数の不動点[1)]」という別の意味付けも与えられる。実は、不動点として捉えることで別アルゴリズムへ変換など停止問題を統一的に扱う助けになる。

さらに、複雑なプログラムを理解するためには、自然数や順序集合では収まらず、量子計算や圏モデルなど、すこし高度な数理モデルが必要になる。“プログラムが住む領域”の発見が、単一のプログラムの理解を超えて、「計算やアルゴリズムとは何か」を解き明かすヒントになる。

ところで、アマゾンの少数民族の“ピダハン語”は、言語学者を驚かしている。例えば「ダンが直す舟を運んでくれ」は「船を運んでくれ。ダンが船を直す。同じ船だ。」と表される。これは、直接見聞きした事実以外は重視しないため、伝聞や推測文を可能とする再帰構造を文法が許さないからだ。この地では、変に空気を読むことなく、人はとてもシンプルになれるのだろう。

（外園 康智）

1）1つの関数をある微分方程式の解として特徴づけることに似ている。