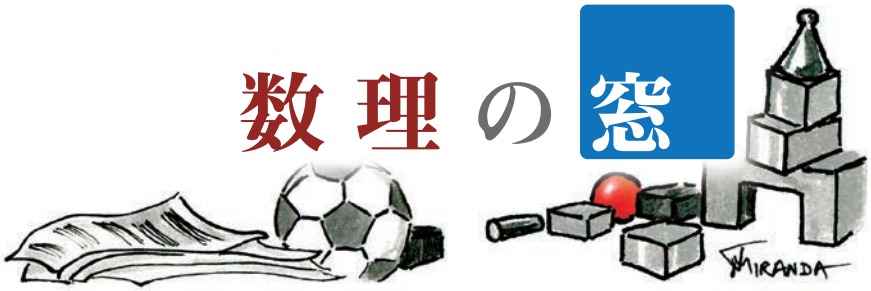
# 数理の窓




## システムを進化させるシステム

```
数字を昇順にソートするプログラム
for i in range(1, len(items)):
    pos = i
    current = items[i]
    while pos > 0 and items[pos-1] > current:
        items[pos] = items[pos-1]
        pos -= 1
    items[pos] = current
```

1つのプログラムが、仕様通り正確に動くことを“証明”するのは難しい。このため、仕様書からバグのないプログラムを自動生成する「自動プログラミング」の研究が進んでいる。「定理証明支援系[1)]」はその1つで、「論理式の証明＝プログラム」という原理を使う方法だ。ステップは、次の通りだ。

①途中や最終結果の“満たすべき性質”の列挙
②定理証明支援系言語による“仕様”の記述
③満たすべき性質の論理式の証明
④通常のプログラム言語[2)]での自動出力

上記のステップを順に追うと、まず①の満たすべき性質は「ソート後のリストは小さい順に整列」と「リストの要素は、ソート前後もすべて同一」の2つだ。後者の論理式は「任意の数字xに対して、ソート前リストに属するxの数と、ソート後リストに属するxの数が等しい」と表現できる。

②は、例えばリスト {3,4,8,11} に、数字7を挿入し {3,4,7,8,11} にしたい場合、順序を保ちつつリストに要素を加える挿入式を作る。さらに、挿入式を再帰的に処理する式を定義する。そして③で、この式に対して①の論理式の証明ができると、仕様通りで、かつバグのないソートプログラムが自動的に出力できる。

定理証明支援系による自動プログラミングは、頭で処理しきれない膨大な組合せや無限を扱う場合に、特に有効となる。ただし、ステップ③の論理式の証明は、一般にNP問題[3)]であり、プログラミングより難しい可能性はある。つまり、論理学者かプログラマーのどちらか一方は苦労するのだ。

システム開発では、テスト工程の負担が最も重い。テストなしでバグチェック、自動コーディングまでできると、開発方法が劇的に変わる可能性がある。

自動プログラミングは、システムがシステムを生み出す世界の予兆だ。“システムが自身より複雑な仕様書を書ける”ならば、まさに進化が始まり、人間を超える日も遠くないだろう。。　（外園 康智）

1）論理学の証明の各ステップは、データと関数の両方を表現するラムダ形式に対応し、その全体がプログラムとなる。カリーハワード対応という。有名なものはCoq, Agda, HOL。四色問題や奇数位数定理への適用がある。
2）OCaml Haskell, Schemeなど。
3）証明が示されれば、その検証は簡単だが、証明自身に決まった手続きがなく難しい問題のクラス。