The Reasoned Schemer 前書き

2018 年 3 月 29 日

Commit ID: 30d7dac

The Reasoned Schemer はたびたび奇想天外なものを探求し、ときには挫折させ、そして常に関係プログラミング（relational programming）の世界の虜にする。

“little” シリーズの最初の本である *The Little Schmer* は、個々のプログラムは数学の関数に対応しているという関数プログラミング（functional programming）のアイディアを示した。簡単な関数の例は *square* で、これはある整数をその整数でかけ合せ、たとえば $square(4) = 16$ となる。一方で *The Reasoned Schemer* はプログラムは数学的な関数を一般化する関係（relation）に対応するという関係プログラミングのアイディアを示す。ある関係 $square^0$ は整数の組によって *square* を一般化する。たとえば $square^0(4, 16)$ は 4 と 16 を関連づける。私たちは引数が与えられた $square^0(4, 16)$ のよ

うな関係をゴール（*goal*）と呼ぶ。ゴールは成功（*succeed*）または失敗（*fail*）、そして値を持たない（*have no value*）である。

square と比べて $square^0$ の大きな利点は柔軟さである。具体的な整数のかわりに未知の値を表現する変数（*variable*）を $square^0$ へ渡すことで、整数とそれらの二乗による問題の多彩さを表現できる。たとえばゴール $square^0(3, x)$ は変数 x に 9 を結び付けて成功する。またゴール $square^0(y, 9)$ は変数 y に -3 と 3 を結び付けて 2 つ成功する。もし関係 $square^0$ を適切に書いたならば、ゴール $square^0(z, 5)$ は失敗する。そして私たちは二乗して 5 となるような整数が存在しないと結論付ける。そうでなければ、ゴールは値を持たず、z に関するいかなる結論も導けない。2 つの変数を利用することで、ゴール $square^0(w, v)$ は最初の整数の二乗が二番目の整数となるような全ての整数の組を列挙するという、無限回の成功となる。また順序を考えない 2 つのゴール $square^0(x, y)$ とゴール $square^0(-3, x)$ は、これらを共に利用することで、x に 9 を結び付け、y に 81 を結びつけることで成功する。ようこそ、関係プログラミングの不思議で素晴しい世界へ！

この本は 3 つのテーマがあり、(1) 関係とゴールをどうやって理解・利用そして作成するか（1-8 章）、私たちを関係プログラミングから非純粋なヴァリアント（variant）へと連れていく非関係（*non-relational*）的な演算子を利用するか（9 章）、そして Scheme 上でどのように完全な関係プログラミ

ング言語を実装するか（10 章、付録 A）。

私たちは *The Little Schemer* のほとんどの章の Scheme で書かれた関数を関係へ翻訳する方法を示す。関係によるプログラミングの力を理解したら、7 章と 8 章でその力をありふれた数学的演算子を関係として定義することに利用する。$+^0$ 関係は足し算ができるだけでなく引き算もでき、$*^0$ はかけ算ができるだけではなく因数分解もでき、そして、log^0 は対数を求めることができるだけではなく対数とある数からその底を求めることもできる。つまり足し算の関係から引き算の関係を定義することができ、対数の関係からべき乗の関係を定義することができる。一般的に $(*^0\ x\ y\ z)$ が与えられたとき、私たちはこれらの数について知っていることを指定できる（これらの値は奇数か偶数かなど）。そして $*^0$ に指定されていない値を探すように依頼できる。この仕事を達成する方法を私たちが指定することはなく、どちらかといえば私たちは結果として何が欲しいかを述べる。

この関係的な思考は計算を理解する別の方法であり、これは小さな低レベル言語を用いて表現される。私たちは 1 章で、関係プログラミングの基本的な記法を導入するためにこの言語を使用し、そしてそれは 10 章の私たちによる実装である。1 章の後は、Scheme の `equal?`や `let`、`cond` や `define` といったわずかにフレンドリーなシンタックスに切り替える。これにより Scheme の関数を関係へ翻訳しやすくなる。次は高レベルなシンタックスである。

```
(≡ t0 t1) (fresh (x ...) g ...) (cond^e (g ...
    ) ...) (defrel (name x ...) g ...)
```

関数 ≡ は 10 章で定義され、`fresh` と $\texttt{cond}^e$ そして、`defrel` は付録の配線（**Connecting the Wires**）で Scheme のシンタックス拡張（syntactic extension）機能を利用して定義される。

関係プログラミングを理解するために唯一必要なことはリストと再帰に精通していることだ。10 章の実装では値としての関数への理解も必要とする。それは、関数は引数にも関数呼び出しの値にもなりえるということである。それだけで、私たちは数学や論理の他に進んだ知識を仮定しない。

私たちは明瞭性を高めるために、記号（punctuation）の表記を曲げた。特に、特殊なシンボルか閉じ右括弧で終わる二段組の左側のクエッションマークは取り除いた。これをすると、たとえばクエッションマークで終わるのか関数名なのかの混乱を回避でき、さらにリストの括弧の散乱を少なくできる。

食べ物（の絵）が例としてこの本のいたるところに出てくる理由は 2 つある。1 つ目は、食べ物が抽象的なシンボルより簡単に可視化できるからだ。私たちは食べ物の絵が読者に例とコンセプトを理解する助けになることを願っている。2 つ目は、私たちは主題がどのように挫折させるのかを知っていて、そして、これらの料理の娯楽は読者の食欲を刺激する。

私たちは食べ物について考えることが読者に読書を止めて何かを食べることを促せたらよいと願っている。

君たちはもうスタートする準備がととのった。Good luck！ この本を楽しめることを願っている。

Bon appétit!

Daniel P. Friedman
Bloomington, Indiana

William E. Byrd
Salt Lake City, Utah

Oleg Kiselyov
Sendai, Japan

Jason Hemann
Bloomington, Indiana

References

[1] Daniel P. Friedman et al. *The Reasoned Schemer (MIT Press)*. second. The MIT Press, Mar. 2018. ISBN:

9780262535519. URL: http://amazon.co.jp/o/ASIN/0262535513/.