GSLetterNeo vol.150

2021 年 1 月

プログラミング言語 Koka 入門

熊澤 努 kumazawa @ sra.co.jp

はじめに

Koka[1]（「こうか）と読みます）は、Microsoft 社で開発されている関数型プログラミング言語です。関数型プログラミング言語は、関数を組み合わせてプログラムを構築するように設計されたプログラミング言語の総称です。Haskell、F#、Racket などの言語が関数型言語の特徴をもつ言語として知られています。

Koka の特徴は「副作用」の扱いにあります（Koka では effect と呼んでおり、この日本語訳である「効果」が言語名に採用されました）。副作用とは、画面への表示、ファイルの読み書き、大域的な記憶領域への状態の保存といった、プログラムの実行に影響を与える処理のことです。例えば、読み込んだファイルの内容によって動作を変えるプログラムを考えると、ユーザがファイルの内容を書き換えることで、プログラムそのものは変更していないにもかかわらず、異なる振る舞いをするようになります。一方で、副作用のないプログラムは、同じ入力に対して同じ結果を常に計算する「書いた通りに動くプログラム」です。Koka は、副作用の情報を「型」として持つことで、扱いやすくすることを目指して開発されました。

今回の記事では、Koka を使って簡単なプログラムを書いてみます。記事に掲載するプログラムは、2020 年 12 月時点で最新の Koka version 2.0.12 で動作を確認しています。

---

[1]詳しい情報は以下の公式サイトを参照してください。
https://www.microsoft.com/en-us/research/project/koka/#:~:text=Koka%20is%20a%20function%2Doriented,every%20function%20is%20automatically%20inferred.

Koka をインストールする

Koka は、Linux や MacOS にインストールするのが簡単です。Windows 10 にインストールする場合には、WSL(Windows Subsystem for Linux)[2]を利用するのがよいでしょう。他の OS の場合と同様の手順でインストールすることができます。ここでは、筆者が利用している WSL（Ubuntu 18.04）にインストールする手順を解説します。その他の環境でのインストール手順についてはプロジェクトの github レポジトリ[3]を参照してください。
WSL を起動して、公式サイトのインストール手順[4]に記載されている下記のコマンドを実行します。なお、Koka を使うには、GCC[5]をあらかじめインストールしておく必要があるので注意してください。GCC のインストール手順は割愛します。

```
> curl -sSL https://github.com/koka-lang/koka/releases/latest/download/install.sh | sh
```

エラーが発生せずにコマンドが終了すれば、インストールは完了です。
確認のために、Koka のインタプリタを起動しましょう。以下のコマンドを実行して、Koka のロゴが表示されれば成功です。

```
> koka
```

初めての Koka プログラミング

Koka 言語でのプログラミングを実際に試してみましょう。Koka にはコンパイラと対話型インタプリタが用意されていますが、今回はコンパイラを使います。
次のプログラムは、文字列 programming language Koka! を画面に出力します。print.kk という名称のファイルとして保存してください。Koka のソースファイルには.kk という拡張子をつける決まりです。

```
fun main() {
    println("programing language Koka!")
}
```

---

[2] https://docs.microsoft.com/ja-jp/windows/wsl/

[3] https://github.com/koka-lang/koka

[4] https://koka-lang.github.io/koka/doc/book.html

[5] https://gcc.gnu.org/

Kokaでは、様々な計算処理を関数で記述します。print.kkは関数mainのみで構成されています。関数の定義の先頭には予約語funを付け、その後に関数名mainを書きます。その後のかっこ()内には関数の引数を書きますが、main関数には引数がないので空欄になっています。関数の範囲は中かっこ{}で定めます。main関数は、C言語のmain関数やJavaのmainメソッドと同様に、プログラム実行後最初に実行される関数と定められており、プログラマは必ず一つ書かなければなりません。main関数内のprintln関数は、引数に指定した文字列を画面に出力するライブラリ関数です。ここでは、文字列 programing language Koka! を画面に出力します。

次に、koka コマンドでソースファイルをコンパイルします。koka コマンドには-c オプションを付けます。

```
> koka -c print.kk
compile: print.kk
loading: std/core
loading: std/core/types
loading: std/core/hnd
check  : print
linking: print
created: out/v2.0.12/gcc-debug/print
```

コンパイルに成功すると、コンパイラの出力の最終行にあるように、実行ファイル（out/v2.0.12/gcc-debug/print）が生成されます。このファイルを実行すると、作成したプログラムを実行することができます。

```
> out/v2.0.12/gcc-debug/print
programing language Koka!
```

コンパイルと実行を一度に行うには、-c コマンドをつけずに koka コマンドを実行すればよいようです。ファイル名を指定するのを忘れないでください。

```
> koka print.kk
compile: print.kk
loading: std/core
loading: std/core/types
loading: std/core/hnd
check  : print
linking: print
created: out/v2.0.12/gcc-debug/print

programing language Koka!
```

クイックソート

もう少し複雑な例として、整列アルゴリズムの一つであるクイックソートを実装します。クイックソートは、値の小さい順、あるいは大きい順に配列やリストを並び替えるアルゴリズムです。ピボットと呼ばれる値より小さい値を集めた部分リストと、大きい値を集めた部分リストを作り、これら二つの部分リストに対して再帰的にクイックソートを実行することで、リスト全体を並び替えます。
整数から成るリスト `[3, 2, 4, 6, 1, 5]` を値の小さい順に、また、文字列 `software` の各文字をアルファベット順に並び替えるプログラムを以下に示します。

```
fun main() {
    [3, 2, 4, 6, 1, 5].quick-sort((<)).show.println // 整数リストの並び替え
    "software".list.quick-sort((<)).string.println // 文字列の並び替え
}

// クイックソート
fun quick-sort(xs, lt) {
    match (xs) {
        Nil -> Nil // 空の場合
        Cons(piv, xss) -> { // 空でない場合、再帰的に並び替える
            var less := xss.filter(fn(x) {x.lt(piv)}).quick-sort(lt)
            var greater := xss.filter(fn(x) {!x.lt(piv)}).quick-sort(lt)
            less + [piv] + greater
        }
    }
}
```

このプログラムを quick-sort.kk というファイル名で保存した後、コンパイルして実行してください。以下のように、整数リストも文字列も正しく並び替えられたことが分かります。

```
> koka quick_sort.kk
compile: quick-sort.kk
loading: std/core
loading: std/core/types
loading: std/core/hnd
check  : quick-sort
linking: quick_dash_sort
created: out/v2.0.12/gcc-debug/quick_dash_sort

[1,2,3,4,5,6]
aeforstw
```

関数 quick-sort から見ていきます。この関数には引数が二つあります。xs は並び替えたいリストで、lt は並び替えの順序を決める値同士の比較関数（小さい順に並べる場合は、どちらの値が小さいかを決定する関数）です。このように、Koka は関数を引数として取ることができます。

予約語 match はパターンマッチングを表し、xs の値のパターンに応じて処理を切り替えます。Koka におけるリストは、Nil または Cons(x, xss)のどちらかの値になります。Nil は空リストを意味します。空リストに対しては、並べ替え後も空リストなので、quick-sort 関数は Nil を返します。Koka は関数の最後に書いた式の値を返します（return Nil としても構いません）。Cons(x, xss)は、空リストではなく、先頭が x でそれ以降が xss であるようなリストを意味します。例えば、リスト [3, 2, 4, 6, 1, 5]の場合には、x が 3、xss が[2, 4, 6, 1, 5]です。

リストが空リストでない場合には並び替え処理を実行します。予約語 var は局所変数の定義です。変数 less にはリストの先頭の値より小さい値（小さい順に並び替える場合）をリストとして格納します。filter 関数は Koka のライブラリ関数で、与えられた条件を満たす値をリスト xs から取り出します。条件は引数として関数 fn(x) {x.lt(piv)}で指定しています。fn は無名関数という名前を付けない関数の定義に使う予約語です。ここでは、ピボット piv の値より小さい値のみを選んでいます。また、Koka の特徴的な記法であるドット記法を併せて使っています。filter 関数の引数はリスト xs と条件を表す関数 f のため、filter(xs, f)と書きます。しかし、Koka は第 1 引数 xs を関数名の前に移動した xs.filter(f)という記法を許しています。他の関数についても同様にドットを使った記法を使うことができます。この記法により、パイプ処理やオブジェクト指向言語に近い書き方ができるようになっています。このプログラムの場合、xss.filter(fn(x) {x.lt(piv)}).quick-sort(lt)は quick-sort(filter(xss, fn(x) {x.lt(piv)}), lt)と同じ意味となります。つまり、リストの先頭の値 piv より小さい値のみを取り出したリストを quick-sort 関数で再帰的に並び替えます。

一方、変数 greater には、xs の先頭の値以上の値を格納したリストを並び替えた結果が格納されます。感嘆符 ! は否定(NOT)演算に相当する関数です。

最後に、less + [piv] + greater という式で、正しい順序で並び替えたリストを生成して返します。+はリストの連結をするライブラリ関数です。

main 関数では、整数リストと文字列を並び替えて画面に表示する処理を行います。ここでもドット記法を使っている点に注意してください。show はリストを文字列に変換するライブラリ関数、list は文字列を文字のリストに変換するライブラリ関数、string は文字のリストを文字列に変換するライブラリ関数です。整数リスト、文字列のどちらの場合も、「リストを並び替えた後、文字列に変換してから画面に表示する」という流れで処理が行われます。

副作用を扱うプログラム

副作用を扱うプログラムも簡単に見ていきます。クイックソートの途中経過を履歴として、実行順とその逆順に表示するプログラムを以下に示します(quick-sort-out.kk とします)。

```
fun main() {
    print-quick-sort()
    print-reverse-quick-sort()
    ()
}

// プログラマが定義する副作用
effect fun history(xs:some<a>(list<a>)): ()

// クイックソート
fun quick-sort(xs, lt) {
    match (xs) {
        Nil -> Nil
        Cons(piv, xss) -> {
            var less := xss.filter(fn(x) {x.lt(piv)}).quick-sort(lt)
            var greater := xss.filter(fn(x) {!x.lt(piv)}).quick-sort(lt)
            var sorted := less + [piv] + greater
            history(sorted)
            sorted
        }
    }
}

// 並び替えの履歴を表示する
fun print-quick-sort() {
    // 並び替えの途中経過を表示する
    with fun history(xss: list<int>) {
        println(xss.show)
    }
    [3, 2, 4, 6, 1, 5].quick-sort((<))
}

// 並び替えの履歴を逆順に表示する
fun print-reverse-quick-sort() {
    // 並び替えの途中経過を逆順に保存する
    var reverse := Nil
    with fun history(xss: list<int>) {
        reverse := Cons(xss, reverse)
    }
    println("Print Reversed History.")
    [3, 2, 4, 6, 1, 5].quick-sort((<))
    // reverseの各値の表示
    foreach(reverse) fn(e) {println(e.show)}
}
```

実行結果は以下の通りです。

```
> koka quick_sort-out.kk
compile: quick-sort-out.kk
loading: std/core
loading: std/core/types
loading: std/core/hnd
check  : quick-sort-out
linking: quick_dash_sort_dash_out
created: out/v2.0.12/gcc-debug/quick_dash_sort_dash_out

[1]
[1,2]
[5]
[5,6]
[4,5,6]
[1,2,3,4,5,6]
Print Reversed History.
[1,2,3,4,5,6]
[4,5,6]
[5,6]
[5]
[1,2]
[1]
```

関数 quick-sort には、並び替えた後のリスト sorted を引数とする history(sorted)を追加しました。これを Effect Handler といい、プログラマが定義した副作用を伴う処理を実行する箇所です。Effect Handler は quick-sort 関数の上に以下のように定義しています。

```
// プログラマが定義する副作用
effect fun history(xs:some<a>(list<a>)): ()
```

副作用の具体的な内容は、クイックソートを呼び出す関数で定義します。履歴を順に表示する print-quick-sort 関数では、次のように with 文を使って、history を表示処理に束縛しています。

```
    // 並び替えの途中経過を表示する
    with fun history(xss: list<int>) {
        println(xss.show)
    }
```

Effect Handler によって、print-quick-sort 関数で、画面表示という副作用をプログラマが指定できるようになりました。

同様に、逆順に表示する print-reverse-quick-sort 関数では、履歴をリストの先頭に追加保存する処理を Effect Handler に定義しています。この結果得られる履歴の逆順リストを reverse としました。

print-reverse-quick-sort 関数のリスト reverse の値を表示する処理は、ライブラリ関数 foreach を使って書いています。

```
    // reverseの各値の表示
    foreach(reverse) fn(e) {println(e.show)}
```

foreach 関数は、Python の for 文に似た機能を提供し、与えられたリストの各要素に対して繰り返し処理を実行します。foreach 関数の引数は、リスト reverse と、そのリストの値を引数で受け取って表示する無名関数 fn(e) {println(e.show)} です。ただし、この例では、第 2 引数である無名関数は引数を書くかっこの外に書いています。この記法は引数書かれた無名関数を見やすくする Koka の機能で、Trailing Lambda と呼びます。

おわりに

今回は、Microsoft 社が開発しているプログラミング言語 Koka の概要を紹介しました。残念ながら、副作用を型として扱い、強い型付けを行うなど、Koka の重要な特徴は触れることができませんでした。この機能の解説は別の機会に譲りたいと思います。
公式サイトには他にもプログラムの例やチュートリアルが掲載されています。興味のある読者は試してみてください。

GSLetterNeo Vol.150
2021 年 1 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　土屋正人
バックナンバー http://www.sra.co.jp/gsletter
お問い合わせ　gsneo@sra.co.jp

〒171-8513 東京都豊島区南池袋 2-32-8