
# 遅延評価

Prepared for: Trenz Pruca, Title
Prepared by: Urna Semper, Job Title
2015年6月30日

## 正格評価と非正格評価

評価方法には「正格評価」と「非正格評価」が存在する。

従来の言語　➡　正格評価
Haskell　　➡　非正格評価

例えば、次のような関数があるとする。

```
--入力した値が2で割り切れたらFalse 割り切れなかったらTrueを返す
isOdd = mod n 2 == 1
```

この関数を基にHaskellの評価について述べていく。
isOdd (1 + 2)　という式を与えた時の評価について、従来の言語の評価（正格評価）は以下のようになる。

1. 部分式 1 + 2　を評価して 3 を得る
2. n に 3 を束縛して isOdd関数に適用する
3. mod 3 2 を評価して 1 を得て、 n == 1 を評価して True を得る

一方で Haskellの評価（非正格評価）では部分式 1 + 2 は値 3 に簡約されない。
そのかわり、式 isOdd (1 + 2) の値が必要になったときにそれを計算できるような「プロミス」を作るのである。この「プロミス」のことを **サンク** と呼ぶ。この式の結果がその後使われる事がないなら、この値が計算される事はない。
非正格な評価は　**遅延評価**　と呼ばれる。

## 遅延評価の性質

遅延評価とは、式の評価を実際に必要になる場面になるまで遅らせる評価方法である。この評価方法をとる事で評価に冗長性がなくなり、演算の効率化を計る事が出来る。他にも、短絡であるが故に持ち合わせる性質もある。

## 短絡であるが故の性質

以下のようなor演算について考えてみる。

if True || length [1..] > 0

非正格な評価では || の左側で条件を満たすようであれば右側の式は計算しない。よって上の式では左側の式 True のみでthen評価に移るのである。ここで、左側の式 length [1..] > 0 は無限リストを計算する式であるのでこれが評価されてしまうと無限ループに陥って抜けるのにプロセスを殺す事になってしまう。
pythonなどの言語で同じように書くと、正格評価で噛み付かれる事になる。

## 評価される必要のない式

最終評価結果がサンクのままの場合もある。例えば、以下のような関数を考える。

```
myDrop n xs = if n <= 0 || null xs -- null関数：値がNULLかどうかBOOLで返す
    then xs
    else myDrop (n-1) (tail xs)
```

myDrop関数では、特定の文字列から任意の文字数頭から取り去って残りを返す。これは、文字列の頭以外を返すtail関数を再帰によって繰り返して実現している。文字列”abcde”を与えたときについて考えてみる。

myDrop 2 “abcde” == tail “bcde”
➡True

ここで、tail “bcde” はサンクとなり必要な場面になるまで評価されない。myDrop 2 “abcde”の評価について順を追って確認してみる。

myDrop 2 “abide”
myDrop (2 - 1) (tail “abcde”)
myDrop 1 (tail “abcde”)
　➡ n <= 0 かどうか評価するためにサンク（2 - 1）を評価

myDrop 1 “bcde”
　➡ null xs かどうか評価するためにサンク（tail “abcde”）を評価

ここまでが再帰の 1 巡目である。続きを見てみる。

myDrop (1 - 1) (tail “bcde”)
　➡n <= 0かどうか評価するためにサンク（1 -1）を評価

myDrop 0 (tail “bcde”)
　➡この時点でmyDrop関数のif文はTrueになりThenを通り、xsが返る

tail “bcde”

ここで、元の式

myDrop 2 “abcde” == tail “bcde”
が再帰適用の結果

tail “bcde” == tail “bcde”
とサンク同士が明らかに等しい状態になった。この時点でtail “bcde”は評価する必要がなくなり、ここで演算は終了するのである。

## 今回のポイント

・Haskell の式の評価を理解するためには置き換えを利用するのが有効

・遅延評価は値が必要になるまで式の評価を遅らせ、また値が必要になった場合も最低限の評価で済ます

・関数適用の結果はサンク（遅延式）になることもある