#21 関数型プログラミングのための参照透過性を保つことのできる高速なリスト

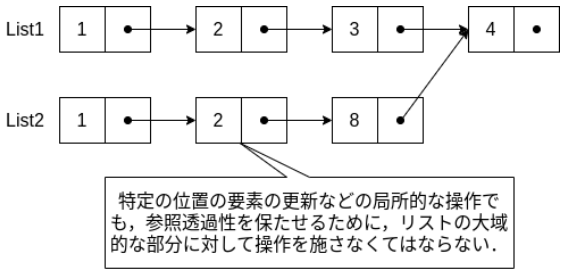
二又 康輔 （久留米工業高等専門学校 3年）

メンター：室屋 晃子（京都大学数理解析研究所 助教）

グローバルサイエンスキャンパス(GSC)
情報科学の達人

関数型プログラミングにおけるリストの問題点

- 手続き型プログラミングにおいてはリストに対する要素の変更は可変長配列を用いると定数時間で行える。
- 関数型プログラミングでは参照透過性を保たせるために、リストの要素を変更するたびに新たなリストを作り直す必要があり、これにはリストの長さに比例した計算量を要する。



目標：参照透過性を保つことのできる高速なリストを作成する。

手法

- 観察：多くのリストに対する操作は最新版に対してのみ行われる。
  最新版へのリストに対する操作が高速であるリストの作成を目指す。
- 考察：リストに関する主要な破壊的操作は適切な入力に関しては可逆である。

以下、適切な入力の場合のみを考える。
例えばリストの特定の位置の要素を変更するupdate操作を以下のように定義する。

$$update(\langle L, i, b\rangle) = \langle L', i, a\rangle$$

ただし、

$$L = x_0 \cdots x_{i-1} a x_{i+1} \cdots x_{n-1}$$
$$L' = x_0 \cdots x_{i-1} b x_{i+1} \cdots x_{n-1}$$

すると

$$update \circ update = id$$

となり、可逆である。
他にもリストの特定の位置に要素を追加するinsert操作と特定の位置の要素を削除するremove操作についても互いに可逆の関係にある。今回はこれら3つの操作について考えた。
以上のことを踏まえて、3つの操作を識別子と操作を実行する関数で表現する。ただし、fstはペアの最初の要素を取り出す関数である。

$$do\left(L, U_i^{a,b}\right) = fst(update(\langle L, i, b\rangle))$$
$$do(L, I_i^a) = fst(insert(\langle L, i, a\rangle))$$
$$do(L, R_i^a) = fst\left(remove(\langle L, i\rangle)\right)$$

上で定義した識別子について操作自体の可逆性に矛盾がないように逆操作を考える。ただし、Sを任意の操作とする。

$$\overline{U_i^{a,b}} = U_i^{b,a} \quad \overline{I_i^a} = R_i^a \quad \overline{R_i^a} = I_i^a \quad \bar{\bar{S}} = S$$

目的のリストを通常のリストLと操作履歴の識別子列Hのペアで表現すると、破壊的操作Sを実行する関数call、操作を一つ戻す関数revは

$$call\left(\langle\langle L, H\rangle, S\rangle\right) = \langle do(L, S), S : H\rangle$$
$$rev(\langle L, S : H\rangle) = \left\langle\langle do(L, \bar{S}), H\rangle, S\right\rangle$$

と表すことが出来る。
ここで

$$do(do(L, S), \bar{S}) = L$$

が成り立つ。
つまり、callとrevは互いに可逆の関係にある。

結果、展望

- リストの長さを100,000で固定してランダムなインデックスへの更新操作を数千回行ったところ、新しいリストは従来のリストに比べておよそ100倍ほど速いことが分かった。（左図）
- しかし、過去のバージョンへのアクセスに関しては、従来に比べて数百から数千倍ほど遅いことが判明した。（右図）