可逆線形探索

2014SE006 青木　崚

2014SE024 家崎　雄太

平成 29 年 7 月 26 日

1　はじめに

可逆計算の歴史は浅い. 非可逆な計算を実行すればエネルギーが消費される. エネルギー消費の点で可逆計算は注目されている. 非可逆な計算により消去されるデータをごみデータとして保存することで可逆性を保証する. 出力にごみデータを追加し, 非可逆な計算により失われるデータ, 制御情報を保存する方法を埋め込みという. 線形探索は探索アルゴリズムの一つであり, 最も基本的な探索アルゴリズムである. 今回の目標としては 2 つある. 一つは Landauer 法（埋め込み）と Bennett 法を用いて,C 言語で書かれた非可逆なプログラムの線形探索 3 種類（単純法, 番兵法, 昇順型）の配列版, スタック版を Janus で可逆シミュレーションすること, もう一つはそれぞれの Janus プログラムの正しさについて示すことである.
今回開発環境としては Janus のインタプリタを用いた，また，情報共有のツールとして Googledrive を利用した.

2　Janus 言語

2.1　Janus とは

Janus は命令型可逆プログラミング言語である．Janus は可逆性を保証する構文規則をもつ．Janus を利用することで可逆なプログラミングを記述することが出来る.
Janus の変数宣言は int 型の変数，int 型の一次元配列または int 型のスタックを定義する．代入をおこなう際は，+=（加算），-=（減算）または^=(排他的論理和) を使用する．ただし代入式の左側に現れた変数は右側に現れてはいけない，すなわち x+=x という計算を行うことが出来ない．代入は変数の値を変える唯一の方法である.

条件式 if then s1 else s2 fi e2 について述べる．式 e1 が真であるとき，文 s1 を実行する．文 s1 を実行後，式 e2 は真でなくてはならない．式 e1 が偽であるとき，文 s2 を実行する．文 s2 を実行後，式 e2 は偽でなければならない.

次に繰り返し式 from e1 do s1 loop s2 until e2 について述べる．最初，式 e1 は真でなくてはならない．後の反復では式 e1 は偽でなければならない．最初に文 s1 が実行される．式 e2 が偽であれば文 s2 を実行しする．式 e2 が真であれば実行を終了する.

データ型のスタック操作について述べる．push(x,s) はスタック s に要素 x を追加し，x を 0 クリアする．pop(x,s) はスタック s から一番上の要素を取り出し，x に格納する.
local はローカル変数に記憶領域を割り当て，変数を式と同じ値で初期化する．local と delocal に囲まれた部分でしかその変数を扱うことができない.

関数を呼び出すには call と uncall を用いる．call は関数を呼び出すことができる．uncall は関数を逆呼び出しすることができる.

2.2 C プログラムからの書き換え規則

可逆線形探索を実装するに当たって，C の式から Janus の式への変換の規則を図 1のようにまとめた.

C
```
int x = 0;
```

Janus
```
local int x = 0
 push(x,g)
delocal int x = 0
```

(a) 変数宣言

C
```
X = 1;
```

Janus
```
push(x,g)
x ^= 1
```

(b) 代入

C
```
if(e1){
   s1
}
 else{
   s2
}
```

Janus
```
if e1 then
   s1
   push(1,g)
else
   s2
   push(0,g)
fi top(g) = 1
```

(c) 条件文

C
```
while(e1){
   s1
}
```

Janus
```
push(1,g)
from top(g) = 1 loop
 s1
 push(0,g)
until !e1
```

(d) 繰り返し文 1

C
```
for(i = 0; e1 i++){
   s1
}
```

Janus
```
push(1,g)
from top(g) = 1 loop
 s1
 i += 1
 push(0,g)
until !e1
```

(e) 繰り返し 2

図 1: C を Janus に書き換える為の規則

図 1aは変数宣言の変換規則である．Janus では変数宣言のとき local int を用いる．delocal int の前に push(x,g) を実行し，x を 0 クリアすることで x = 0 が成り立つ.

図 1bは代入の書き換え規則である．C の様な代入は情報が失われるため，Janus では行うことが出来ない．Janus で代入の働きを担う^=は x が 0 であるときのみ実行することができる．そのため, まず push(x,g) を実行し，x を 0 クリアする．そうすることで，x の値に左右されること無く代入を実行することができる.

図 1cは条件文の書き換え規則である．二章で記述したように，条件文のアサーションは if 節を実行したときは真であり，else 節を実行したときは偽でなければならない．そのため，if 節を実行したとき最後に push(1,g) を実行する．そうすることで，アサーションの top(g) = 1 が必ず真になるようにしている．また，else 節を実行した後，最後に push(0,g) をする．そうすることで，アサーションが必ず偽になるようにしている.

図 1dは while による繰り返しの書き換え規則である．C の while 文の条件 e1 を Janus では until に e1 の否定として記述する.

Janus の loop のアサーションは最初に真でなければならない．そこで，push(1,g) を実行し，する．そうすることで，アサーションの top(g) = 1 が必ず成り立つ．また，最初以外のアサーションは偽でなければならない．そこで loop 節の最後に push(0,g) を行う．そうすることで，アサーションが偽になるようにしている.

図 1eは for 文による繰り返しの書き換え規則である．Janus では繰り返しを行うものは loop 文しかないため，基本的に while による繰り返しの書き換え規則と同じ構成である．

3 Janus による可逆線形探索の実装

この章では三種類の C でかかれた線形探索を書き換え規則（図 1）を用いて Janus に書き換え，可逆線形探索の実装を行った．本稿では配列を用いた可逆線形探索とスタックを用いた可逆線形探索の 2 種類の実装を行った．スタックを用いた線形探索の実装に用いられる C プログラムの構造体または関数は図 2にまとめて記述した．また，スタックを用いた線形探索の texttttC プログラムではポインタによる操作をスタックを二つ使うことで，再現している．それは，Janus プログラムに変換する際に，二言語間で対応がとれるようにするためである．

```
#define SMAX 1024

struct stack{
    int box[SMAX];
    int top;
};
```

(a) 構造体 stack

```
void create(struct stack *s)
{
    s->top = -1;
}
```

(b) 関数 create

```
void push(int *x, struct stack *s)
{
    s->top = -1;
    s->box[s->top] = *x;
    *x = 0;
}
```

(c) 関数 push

```
void pop(int *x,struct stack *s)
{
    *x = s->box[s->top];
    s->top -= 1;
}
```

(d) 関数 pop

```
void empty(struct stack *s)
{
    return(s->top == -1);
}
```

(e) 関数 empty

```
void top(struct stack *s)
{
    return s->box[s->top];
}
```

(f) 関数 pop

図 2: C でスタックを実装するための関数

3.1 標準型

標準型はもっとも基本的な線形探索である．前提として配列には必ず数が格納されているとする ($N \geqq 1$)．標準型は探索している配列に数が格納されている確認 ($i \leqq N$) する必要がある．配列を先頭から探索していき，探している値 (以後 key とする) が見つかれば，探索は成功として終了する．また key が見つからず，配列に数が格納されていなければ，探索は失敗として終了する．配列を用いた通常型の C と Janus のプログラムを図 3に記述した．またスタックを用いた通常型の C と Janus のプログラムを図 4に記述した．

3.2 番兵法

番兵法は配列の最後に key と同じ値（ダミー）を格納する．標準型とは異なり，配列に数が格納されているかどうか確認する必要がなくなる．そのため，標準型より計算が早くなる．key と同じ値を比較しているとき，それがダミーでなければ，線形探索は成功として終了する．また，ダミーであれば，線形探索は失敗として終了とする．配列を用いた番兵法の C と Janus のプログラムを図 5に記述した．またスタックを用いた番兵法の C と Janus のプログラムを図 6に記述した．

3.3 並んでいるもの（昇順型）

昇順型は昇順に値が並んでいる配列を探索する．比較している値 $(K_i(1 \leqq i \leqq N))$ が key を超えた時点で線形探索が成功か失敗かを判断する．$K_i \geqq$ key のとき $K_i =$ key であれば線形探索は成功として終了する．また，$K_i! =$ key であれば失敗として終了する．配列を用いた昇順型の C と Janus のプログラムを図 7に記述した．またスタックを用いた昇順型の C と Janus のプログラムを図 8に記述した．

```
int linear(int k[],int key,int n,int f)
{
     int i;

     for(i = 0;(i < n) && (f == -1);i++){
        if(k[i] == key){
            f = 1;
         }
       }
        return f;
}
```

(a) C プログラム

```
procedure linear(int k[],int key,int n,int f,stack g)
    local int i = 0

      push(1,g)
      from top(g) = 1 loop
         if k[i] = key then
             push(f,g)
             f ^= 1
             push(1,g)
         else
             push(0,g)
         fi top(g) = 1
             i += 1
             push(0,g)
      until i >= n || f != -1

      push(i,g)

    delocal int i = 0
```

(b) Janus プログラム

図 3: 配列を用いた標準型のプログラム

```
int slinear(struct stack *s,
            int key,struct stack *t)

    int f = -1;
    int x;

    while(!empty(s) && f == -1)
        if(top(s) == key)
            f=1;

        pop(&x,s);
        push(&x,t);

    return f;
```

(a) C プログラム

```
procedure slinear(stack s,int key,
                  stack t,int f,stack g)
    local int x = 0

     push(1,g)
     from top(g) = 1 loop
        if top(s) = key then
            push(f,g)
            f ^= 1
            push(1,g)
        else
            push(0,g)
        fi top(g) = 1
        pop(x,s)
        push(x,t)
        push(0,g)
     until f != -1 || empty(s)

     push(x,g)
   delocal int x = 0
```

(b) Janus プログラム

図 4: スタックを用いた標準型のプログラム

```
int banpei(int k[], int key, int n)
{
     int i = 0;
     int f;

     k[n] = key;

     while (k[i] != key) {
          i++;
    }

     if (i < n){
         f = 1;
     }
     else{
         f = -1;
     }

     return f;
}
```

(a) C プログラム

```
procedure banpei1(int k[],int key,int n,int f,stack g)
    local int i = 0

      push(k[n], g)
      k[n] ^= key

      push(1,g)
      from top(g) = 1 loop
         i += 1
         push(0,g)
      until k[i] = key

      if i < n then
          push(f,g)
          f ^= 1
          push(1,g)
      else
          push(f,g)
          f ^= -1
          push(0,g)
      fi top(g) = 1

      push(i,g)
    delocal int i = 0
```

(b) Janus プログラム

図 5: 配列を用いた番兵法のプログラム

```
int banpei(struct stack *s,
           int key,struct stack *t)
{
     int temp = key;
     int x;
     int f = 0;
     push(&temp, t);

     while(!empty(s)){
         pop(&x, s);
         push(&x, t);
     }


     while(top(t) != key){
         pop(&x, t);
         push(&x, s);
     }
     pop(&x, t);
     push(&x, s);

     if(!empty(t)){
         f = 1;
     }
     else{
         f = -1;
     }

     return f;
}
```

(a) C プログラム

```
procedure banpei(stack s,int key,
                 stack t,int f,stack gb)
    local int x = 0

      local int temp = key
        push(temp, t)
      delocal int temp = 0

      push(1,gb)
      from top(gb) = 1 loop
         pop(x,s)
         push(x,t)
         push(0,gb)
      until empty(s)

      push(1,gb)
      from top(gb) = 1 loop
         pop(x,t)
         push(x,s)
         push(0,gb)
      until top(t) = key

      pop(x,t)
      push(x,s)

      if !empty(t) then
          push(f,gb)
          f ^= 1
          push(1,gb)
      else
          push(f,gb)
          f ^= -1
          push(0,gb)
      fi top(gb) = 1

      push(x,gb)
    delocal int x = 0
```

(b) Janus プログラム

図 6: スタックを用いた番兵法のプログラム

```
int aolinear(int k[],int key,int n,int f)
{
     int i = 0;

     while (key > k[i]){
           i++;
     }

     if (k[i] == key && i != n){
         f = 1;
     }
     else{
         f = -1;
     }

     return f;
}
```

(a) C プログラム

```
procedure aolinear1(int k[],int key,int n,int f,stack g)
    local int i = 0

      push(1,g)
      from top(g) = 1 loop
         i += 1
         push(0,g)
      until key <= k[i]


      if k[i] = key && i != n then
          push(f,g)
          f ^= 1
          push(1,g)
      else
          push(f,g)
          f ^= -1
          push(0,g)
      fi top(g) = 1

      push(i,g)
    delocal int i = 0
```

(b) Janus プログラム

図 7: 配列を用いた昇順型のプログラム

```
int saolinear(struct stack *s,
              int key,struct stack *t)
{
     int x;
     int f = -1;

     while(top(s) < key){
         pop(&x,s);
         push(&x,t);
     }
     if(top(s) == key){
         pop(&x,s);
         push(&x,t);
         if(!empty(s)){
             f = 1;
         }
     }

     return f;
}
```

(a) C プログラム

```
procedure saolinear(stack s,int key,
                    stack t,int f,stack g)
    local int x = 0

      push(1,g)
      from top(g) = 1 loop
         pop(x,s)
         push(x,t)
         push(0,g)
      until top(s) >= key

      if top(s) = key then
          pop(x,s)
          push(x,t)
          if !empty(s) then
              push(f,g)
              f ^= 1
              push(1,g)
          else
              push(0,g)
          fi top(g) = 1
          push(1,g)
      else
          push(0,g)
      fi top(g) = 1

      push(x,g)
    delocal int x = 0
```

(b) Janus プログラム

図 8: スタックを用いた昇順型のプログラム

4　おわりに

今回の目標は C で書かれた非可逆なプログラムの線形探索 3 種類の配列版とスタック版を Janus で実装すること. それぞれのプログラムの正しさについてしめすこと, この二つである. 一つ目の実装については 6 種類それぞれのプログラムをまず C で書き,Janus で埋め込み,Bennett 法を用いて書き換えた.Janus のインタプリタ Playground で実行できるものを作成できたので, 目標は達成できたといえる. プログラムの正しさについては, それぞれのプログラムのフロー図を作成し条件網羅テストを行った. それぞれ 6 種類のプログラムは入力に対して望まれる結果を返した. 正しさについて目標を達成できたといえる.

開発環境に関しては二人で開発を行った為不自由は無かったが，さらに大人数で行うとなるとバージョン管理の点で不自由を感じた．github を利用して開発を行うべきであった.

成果物は github(https://github.com/yokoyama-lab/reversible-linear-search) にアップロードした.

5　参考文献

1)Holger Bock Axelsen, Tetsuo Yokoyama：Programming Techniques for Reversible Comparison Sorts.
2)Tetsuo Yokoyama：Reversible Computation and Reversible Programming Languages. C.H.Bennett：Logical Reversibility of Computation.
3)DONALD E.KNUTH：The Art of Computer Programming Volume 3 Sorting and Searching Second Edition 日本語版.
4)T. コルメン C. ライザーソン R. リベスト C. シュタイン：世界標準ＭＩＴ教科書アルゴリズムイントロダクション第３版　近代科学社.