実践離散数学と計算の理論 5/31 課題

28G23027
川原尚己

問１：チューリングマシンについて自分なりにまとめて解説せよ．

チューリングマシンとはテープ・ヘッド・状態機械・状態遷移テーブルからなる仮想的な機械である．状態機械のヘッドは現在の状態及びヘッドが指している文字を入力として，状態遷移テーブルに従って次の状態，ヘッドが指す先に印字すべき文字，ヘッドの動かすべき方向を決定する．

チューリングマシンの実行結果の実行列の意味は，ステップiの状態機械S_i，ヘッドh_i，テープw_iの組をある値p_iに符号化し，この符号化列$(p_0, \dots, p_i)$を符号化することによって実行列全体の符号を取得することができる．

問２：ある高校生向けの科学雑誌に「停止問題について」と記事を書くことになったと仮定して、停止問題について解説せよ。

プログラムの記述では， for 文や while 文の繰り返し処理などが存在する．この繰り返し文の処理内容によっては永遠に処理が終了しない（無限ループ）に陥ってしまう可能性がある．例えば，変数iを$i=1$とし，while 文を繰り返すたびにiに1を加算し，iが0未満になれば終了するというような処理を考えてみよう（図１）．

```
i = 1
while i >= 0:
    i += 1
```

図１　処理の python コード

この処理は当然のことながら終了することがない．iの初期値が正であるにもかかわらず，iは増加していく一方であるからである．今回は非常に簡単な例を挙げたが，実際の実行上ではもっと複雑な処理を実行することがある．実行するときまで無限ループがわからないのは非常に不便で危険性があるため，実行前に処理が本当に有限の時間で終了するのか確かめる方法があれば好ましい．このように，「処理が無限ループに陥るかどうか判断できる関数が必ず存在するか」という命題のことを「停止問題」という．さて，一見すると難解そうな問題であるが，実はすでに解決されている問題である．結論から言うと，停止問題の解は「存在しない」である．以下に証明を述べる．

まず，ある計算が停止するか判定することができる halt という関数が定義可能であると仮定する．このとき，入力xに対して，halt(m0, x)=true となるときに，無限ループとなるような関数$m0$を考える．もし，halt(m0, x)=true であると仮定すると，halt は m0 が停止すると判定するが，m0 は実際には無限ループに陥る（停止しない）ので矛盾が発生する．また，halt(m0, x)=false であると仮定すると，halt は m0 が停止しないと判定するが，m0 は実際には終了するのでこちらも矛盾が発生する．どちらの場合でも矛盾が発生するため，halt が定義可能という仮定が誤っていたことになり，計算が停止するか判定可能な関数は存在しないという結論が導かれる．最後に，図２に halt と m0 の疑似コードを示す．

```
halt?(fun, x) {
    if (fun(x)は停止する) {
        return true;
    } else {
        return false;
    }
}
```

```
m0(x) {
    if (halt?(m0, x) = true) {
        loop; /* 無限ループ */
    } else {
        return true;
    }
}
```

図２　関数 halt 及び m0 の定義

問３：Coq を用いて原始帰納的関数を２つ以上、自由に定義せよ.

図３の”pred”，”mul”関数を定義した.

pred 関数は，引数で与えられた自然数の１つ前の数を返す関数であり，

$$pred(x) = \begin{cases} 0, & (x = 0) \\ x - 1, & (x \neq 0) \end{cases}$$

となる．また，mul は２つの自然数の「積」を返す関数である．new_plus 関数は，mul 関数の定義の中に自然数の「和」に関する関数が必要であるため，サンプルコードを mul 関数の型に合うように書き換えたものである.

```
Definition pred (n:nat) : nat :=
  match n with
  | 0 => 0
  | S n' => n'
  end.

Fixpoint new_plus(n:nat) (m:nat): nat :=
  match n with
  | 0 => m
  | S n' => S (new_plus n' m)
  end.

Fixpoint mul(n m:nat) : nat :=
  match n with
  | 0 => 0
  | S n' => new_plus m (mul n' m)
  end.
```

図３　実装関数

各関数の実行例を以下の図 4-7 に示す.

i.　Compute pred 4.

```
     = 3
     : nat
```

図４　Compute pred 4.の実行例

ii.　Compute pred 0.

```
     = 0
     : nat
```

図５　Compute pred 0.の実行例

iii.　Compute mul 3 2.

```
     = 6
     : nat
```

図 6　Compute mul 3 2.の実行例

iv.　Compute mul 0 3.

```
= 0
: nat
```

図 7　Compute mul 0 3.の実行例

これらの関数の定義は，以下の資料を参考とした：
https://www.fos.kuis.kyoto-u.ac.jp/~igarashi/class/cal/handout1.pdf