1J-09

区間演算ライブラリ MPFI への Haskell バインディングの設計と実装

長濱 ななみ† 川端 英之‡ 弘中 哲夫‡
広島市立大学情報科学部情報工学科† 広島市立大学大学院情報科学研究科‡

1 はじめに

精度が保証された計算は様々な場面で必要となる．正確な計算を簡単に行うための手法の一つとして，区間演算ライブラリである MPFI (Multiple Precision Floating-Point Interval Library) [1] が利用される．しかし，Haskell を用いたプログラムにおいて，MPFI を利用することは，メモリ管理が煩雑となるため，容易とは言い難い．一方，任意精度浮動小数点演算ライブラリである MPFR (Multiple Precision Floating-Point Reliable Library) [2] は hmpfr [3] と呼ばれるバインディングにより，Haskell を用いたプログラムでの利用が容易になっている．本研究では，hmpfr の実現手法を踏襲し，MPFI を Haskell プログラムから直接利用可能にするためのバインディングである hmpfi を開発した．本稿では，hmpfi の実現方法について述べる．

2 MPFI の概要

MPFI は C 言語で記述された区間演算ライブラリである．区間は，上限と下限の 2 つの数値で表現される．上限と下限は，任意長仮数部の浮動小数点表現である MPFR のデータ型の数値で表現される．MPFI のメモリ領域は，特別な初期化関数を用いて初期化する必要がある．また，クリア関数を用いてメモリ領域を解放する必要がある．初期化されたメモリ領域には何度でも値を代入や，仮数部長の変更 を行うことが出来る．

MPFI は，以上のように，ユーザが変数のメモリ管理をする必要があるので，手間がかかる．また，MPFI は「手続き」であり，純粋関数型言語である Haskell で使用するのは煩雑である．

3 hmpfi の設計

hmpfi は，図 1 のような API 仕様に整える．

add は, 加算を行う関数，fromString は，値を MPFI 型に変換する関数，outString は，値を文字列として出力する関数である．

MPFI でユーザが行っていたメモリの初期化や解放については，hmpfi が行うようにし，ユーザの記述を容易にする．ライブラリ階層は，MPFI ライブラリをベースとして構築する．hmpfi から MPFI に定義された C 言語の関数を利用可能にするために，FFI(Foreign Function Interface) を用いた．FFI とは，あるプログラミング言語から他言語で定義された関数などを利用するための機構である．

```
add         ::  Precision -> MPFI -> MPFI -> MPFI
fromString  ::  String -> Precision -> MPFI
outString   ::  Word -> MPFI -> String
```

図 1 hmpfi の API 仕様

```
typedef struct {
  __mpfr_struct left;
  __mpfr_struct right;
}__mpfi_struct;
```

図 2 __mpfi_struct 型の構造 (MPFI ライブラリ)

```
typedef struct {
  mpfr_prec_t  _mpfr_prec;
  mpfr_sign_t  _mpfr_sign;
  mpfr_exp_t   _mpfr_exp;
  mp_limb_t   *_mpfr_d;
} __mpfr_struct;
```

図 3 __mpfr_struct 型の構造 (MPFR ライブラリ)

3.1 hmpfi のデータ型

hmpfi を実現するには, hmpfi で扱うデータ型を MPFI のデータ型とやり取りが可能なものにする必要がある．また，MPFI ライブラリのデータ型には MPFR ライブラリのデータ型が使われている．本節では，hmpfi, MPFI ライブラリ, MPFR ライブラリのデータ型について述べる．

図 2 に, MPFI ライブラリで扱うデータ型である__mpfi_struct 型の詳細を示す．

図 3 に，MPFR ライブラリで扱うデータ型である__mpfr_struct 型の詳細を示す．図 3 に示す通り，__mpfr_struct 型は，仮数部長 (_mpfr_prec)，符号 (_mpfr_sign)，指数部 (_mpfr_exp)，仮数部へのポインタ (_mpfr_d) を持つ浮動小数点表現の構造となっている． __mpfi_struct 型は，構造体の中に構造体が格納されている形になっているので，__mpfr_struct 型のフィールド領域に対して，Haskell 側では直接参照が出来ない．そこで，Haskell 側で直接参照出来るように，フィールドの境界を揃えた構造体である__nagahama_struct 型を新たに定義して用いた．Haskell 側では，__nagahama_sturct 型に基づいて，peek/poke により，__mpfi_struct 型のフィールドを参照する．図 4 に， __nagahama_struct 型の詳細を示す．

hmpfi で扱うデータ型は MPFI ライブラリで扱うデータ型に基づいた MPFI 型となっている．図 5 に， MPFI 型の詳細を示す．

MPFR 型は，MPFR ライブラリで扱うデータ型に基づいている．図 6 に， MPFR 型の詳細を示す．

Design and Implementation of a Haskell Binding to the MPFI Library

Nanami Nagahama† Hideyuki Kawabata‡ Testuo Hironaka‡
†Department of Computer and Network Engineering, Hiroshima City University
‡Graduate School of Information Sciences, Hiroshima City University

```
typedef struct {
  mpfr_prec_t _mpfr_prec_l;
  mpfr_sign_t _mpfr_sign_l;
  mpfr_exp_t _mpfr_exp_l;
  mp_limb_t *_mpfr_d_l;
  mpfr_prec_t _mpfr_prec_r;
  mpfr_sign_t _mpfr_sign_r;
  mpfr_exp_t _mpfr_exp_r;
  mp_limb_t *_mpfr_d_r;
}__nagahama_struct;
```

図 4　__nagahama_struct 型の構造 (hmpfi)

```
data MPFI = MP2{ leftM :: MPFR,
 rightM :: MPFR } deriving (Show)
```

図 5　MPFI 型の構造 (hmpfi)

```
data MPFR = MP{ precM :: CPrecision,
 signM :: Sign,
 expM :: Exp,
 limbM :: !(ForeignPtr Limb) } deriving (Show)
```

図 6　MPFR 型の構造 (hmpfi)

```
import qualified Data.Number.MPFI as M
main = do
  let a = M.fromString "1.234567" 100
  let b = M.fromString "9.876543" 100
  let c = M.add 100 a b
  putStrLn $ "show :" ++ c
```

図 7　hmpfi に用意した API の使用例

3.2　hmpfi の内部仕様

hmpfi におけるメモリ管理方式は，hmpfr の実現手法を踏襲する．ここでは関数 add の挙動を例にとして用い，hmpfi によるバインディングの様子を説明する．

1. 引数を，MPFI ライブラリで扱えるように alloc 関数で作業領域を確保し，必要な情報を poke して書き込む．
2. 戻り値を格納する領域を確保し，必要な情報を poke して書き込む．MPFR 型の仮数部は mallocForeignPtrBytes を用いて領域を確保する．
3. 作業領域に格納したデータを引数として，mpfi_add を呼び出す．
4. 計算結果，すなわち MPFI ライブラリ側で値が埋め込まれた作業領域を，hmpfi が peek して読み取り，MPFI 型のオブジェクトを構成し，呼び出し元に返す．

以上の手続きにおいて，作業領域は，add 関数の処理が終わった時点で自動的に解放されるが，仮数部領域はそのまま保持される．

4　hmpfi の実装

hmpfi に用意した API の使用例を図 7 に示す．

fromString 関数はそれぞれ文字リテラルで与えられた数値に基づき，仮数部長 100[bit] を持つ MPFI 型オブジェクトを生成している．add 関数では，仮数部長 100[bit] で確保した領域に加算結果を格納している．hmpfi では，ユーザがメモリの初期化や解放をする必要がなく，MPFI ライブラリを簡便に使えるようになっている．

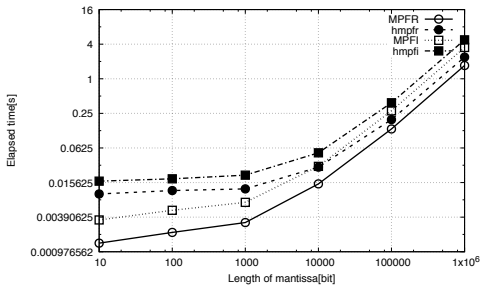

図 8　hmpfi, MPFI, hmpfr, MPFR で計算 (a=a+b, 初期値 a=1.234567,b=9.876543) を 100000 回行った合計時間 (秒)．縦軸は底を 2 とした対数目盛り．

5　評価

hmpfi, MPFI, MPFR, hmpfr のそれぞれのライブラリで計算速度を実測した．評価は，Intel Core i75-3520M(OS X EI Capitan 10.11.6,8GB) の CPU で，ソフトウェアは GHC 7.10.3, MPFR 3.1.5, MPFI 1.5.1, hmpfr 0.4.2 を用いて行った．なお，コンパイルについてはコンパイルオプション -O3 を使用した．

図 8 に，hmpfi, MPFI, MPFR, hmpfr のそれぞれのライブラリで実測した結果を示す．

MPFI の計算時間は MPFR の約 2 倍であることが分かっており，hmpfi の計算時間の差は hmpfr の約 2 倍であることが予想される．図 8 より，実測結果も同様となっていることが分かる．また，hmpfr の計算時間は MPFR の約 1.4 倍であることが分かっており，hmpfi の計算時間は MPFR の約 1.4 倍であることが予想される．図 8 より，実測結果も同様となっていることが分かる．従って，hmpfi は十分効率的な実装が出来ているといえる．

6　まとめと今後の課題

本研究により，Haskell のプログラムにおいて MPFI ライブラリが直接利用可能となった．今後の課題としては，メモリ領域を再利用可能にする方式を API に組み込むことで，hmpfi の更なるパフォーマンスの向上を図ることが挙げられる (予備実験で効果は確認済み)．また，本研究で使用したメモリ管理方式を，我々が開発している実数計算ライブラリ IFN[4] に組み込むことで，IFN の高速化を目指すことも今後の課題といえる，

参考文献

[1] MPFI: https://perso.ens-lyon.fr/nathalie.revol/software.html
[2] MPFR: http://www.mpfr.org/
[3] hmpfr: https://hackage.haskell.org/package/hmpfr
[4] Hideyuki Kawabata and Hideya Iwasaki: Improving Floating Point Numbers: A Lazy Approach to Adaptivr Accuracy Refinement for Numerical Computations, Proc. ESOP 2016, LNCS 9632, pp.390-418, 2016.