Continuation based C による Red Black Tree の Hoare Logic を用いた検証

175706H 上地悠斗 - 琉球大学：並列信頼研究室

背景

- OSやアプリケーションの信頼性を高める事は重要な課題である。
  - 信頼性を上げるために仕様の検証が必要
- 研究室でCbCという言語を開発している。
  - CbCはサブルーチンとループ制御をcから取り除いている。その為、それを実装した際のプログラムを検証をする必要がある。
- CbCは実行を継続してプログラムが動く。この際に正確に動作するのか検証を行いたい。

研究目的

- プログラムの正当性を証明するために Hoare Logic という検証手法がある。

- Hoare Logic とは「プログラムの事前条件(P)が成立しているとき、コマンド(C)を実行して停止すると事後条件(Q)が成り立つ」というものである。
  - これがCbCの実行を継続するという性質と相性が良い

- 本研究ではCbCにおける Red Black Tree の検証を Hoare Logic を用いて行う。

CbCとは

- CbCとは、Cからループ制御構造とサブルーチンコールを取り除き、継続を導入したCの下位言語である。継続呼び出しは引数付き goto 文で表現される

- Code Gear / Data Gear という単位で実装を行う
  - CodeGear は Input DataGear を受け取り、処理を行って Output DataGear に書き込む
  - Output の DataGear は次の CodeGear の Input として接続される

Hoare Logicについて

- 「プログラムの事前条件(P)が成立しているとき、コマンド(C)を実行した後に事後条件(Q)が成り立つ」というものである。
  - 関数が任意の値が引数として実行された際に、任意の値が帰ってきた場合を「関数は正常に実行される」といえる。
- CbC では実行を継続するため、ある関数の実行結果は事後条件になるが、その実行結果が遷移する次の関数の事前条件になる
  - これを繋げていくことで、個々の関数の正当性を証明することと健全性について証明するだけでプログラム全体の検証を行うことができる。

Agdaについて

- Agdaとは定理証明支援器である。
- 型は：で、値は ＝ で与える。
- 構成要素は主に関数, recode, dataの3種類

```
01: record _∧_ (A B : Set) : Set where
02:   field
03:     front : A
04:     back  : B
05:
06: syllogism : {A B C : Set} → ((A → B) ∧ (B → C)) → (A → C)
07: syllogism x a = _∧_.back x (_∧_.front x a)
```

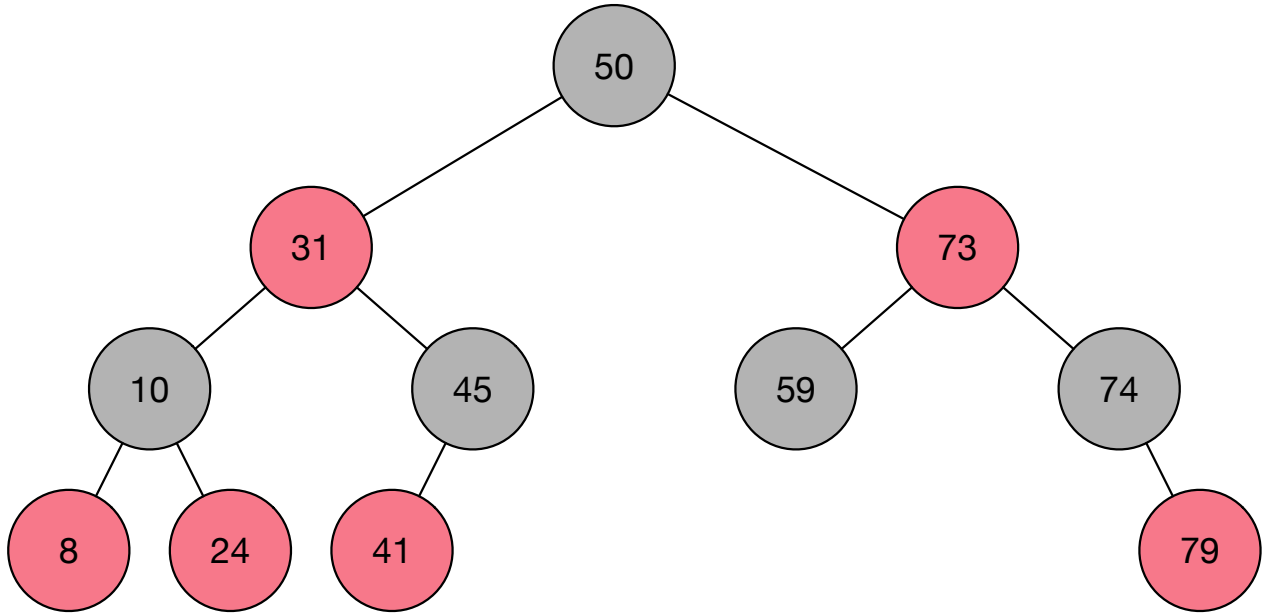
Continuation な Agda の実装

- Agdaでは関数の再帰呼び出しが可能であるが、CbCでは値が帰って来ない。そのためAgdaで実装を行う際に再帰呼び出しを行わないような実装をする。

```
01: plus-com : {l : Level} {t : Set l} → Env → (next : Env → t) → (exit : Env → t) → t
02: plus-com env next exit with varx env
03: ... | zero  = exit (record { varx = varx env ; vary = vary env })
04: ... | suc x = next (record { varx = x        ; vary = (suc (vary env)) })
05:
06: {-# TERMINATING #-}
07: plus-p : {l : Level} {t : Set l} → (env : Env) → (exit : Env → t) → t
08: plus-p env exit = plus-com env ( λ env → plus-p env exit )  exit
09:
10: plus-init : ℕ → ℕ → Env
11: plus-init x y = plus-p (record { varx = x ; vary = y }) (λ env → env)
12: test1 = plus-init 3 4
```

Red Black Tree

- Red Black Tree とは、並行二分木の一つであり、定義は以下となる。
  - 各点は赤か黒の色である。
  - 点が赤である場合の親となる点の色は黒である。
  - 外点 (葉。つまり一番下の点)は黒である。
  - 任意の点から外点までの黒色の点はいずれも同数となる。



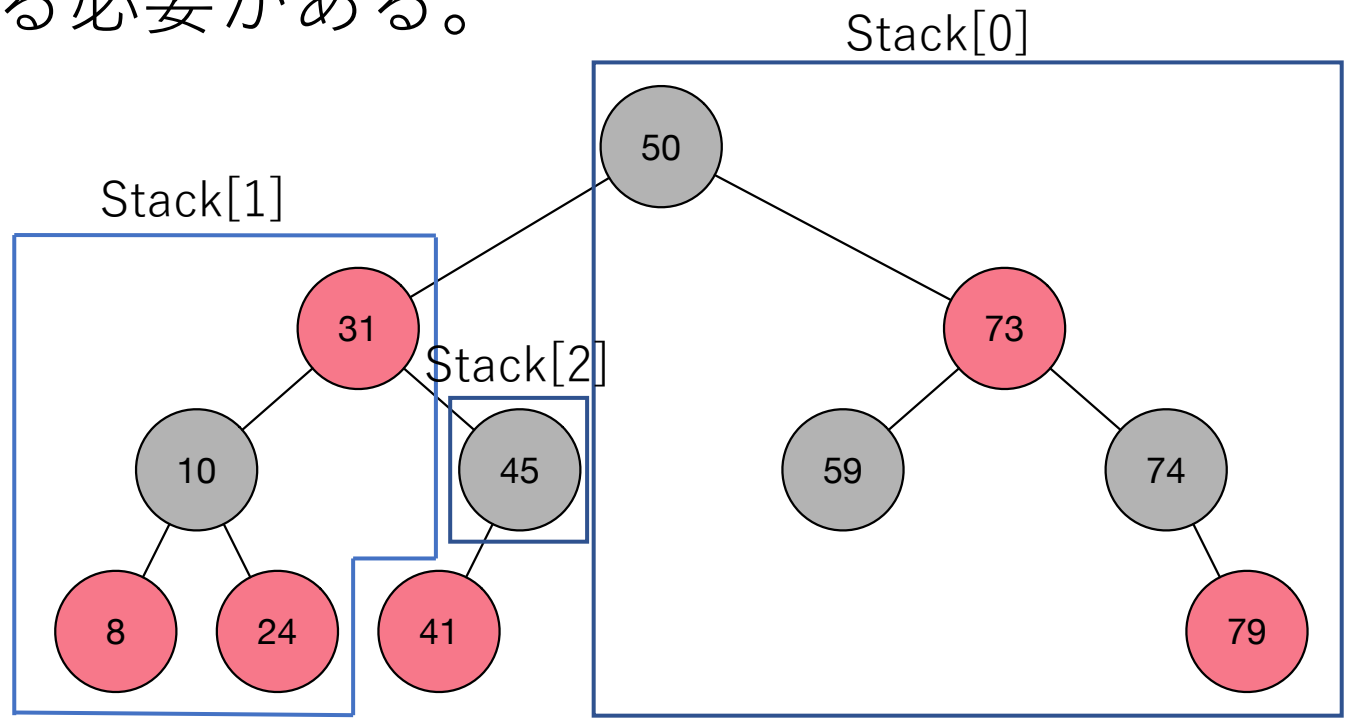
Red Black Tree を検証する目的

- 先行研究では While Loop を検証していた。
- 再帰処理と再代入が必要となるプログラムを検証したい。
- 一般的なデータ構造として知られているRed Black Treeを対象とする。
- Red Black Tree の実装は困難であるため、定義を満たしたアルゴリズムである Left Learning Red Black Tree を検証する。

Red Black Tree の実装

- Agdaは変数への再代入を許していない。そのため、Red Black Tree の実装には一手間加える必要がある。

- Tree へ insert やdeleteをする際には、目的の場所へ辿るまでのnodeを Stack に保持し、実行後に復元する

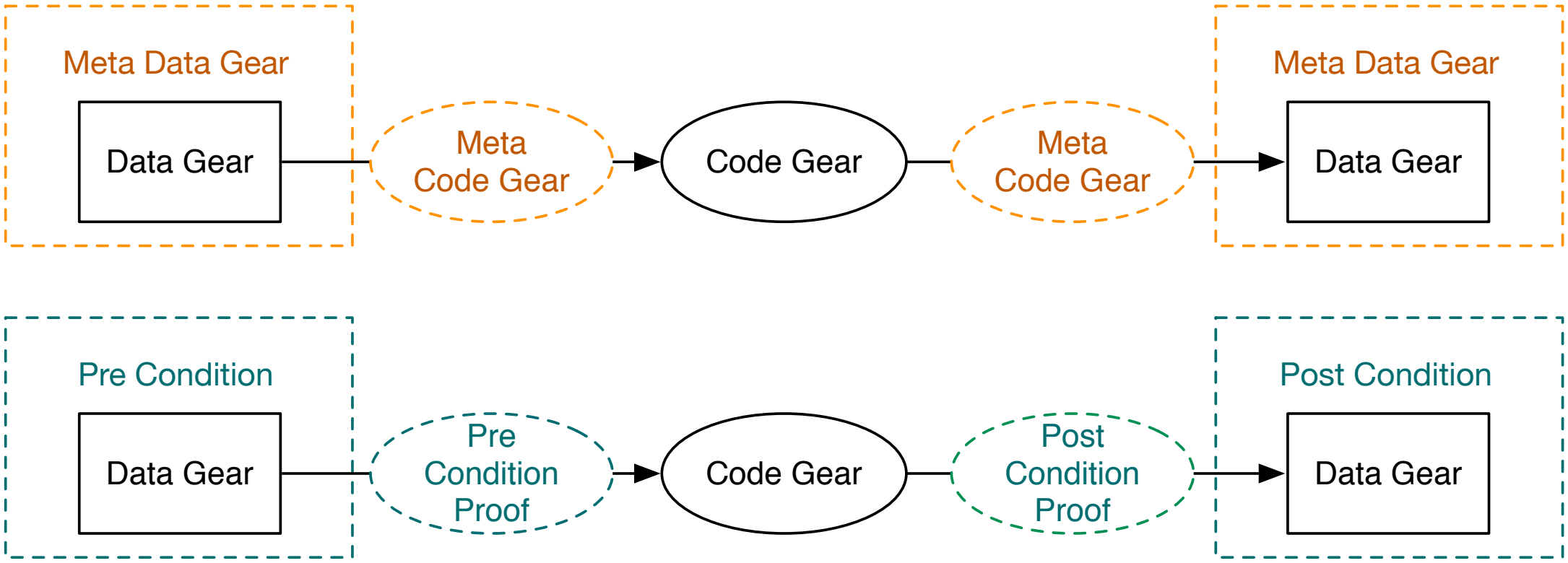


検証

- Input Data Gear が Pre Condition を、 Output Data Gearが Post Condition を 満たしているか検証することで Hoare Logic に当てはめる。

- Meta Data Gearの作成中。
  - Node の大小関係を持った Stack の作成
  - Tree の 深さ、黒ノードの数を保存
  - コマンド毎の Meta Data Gear の作成は未完成
  - Pre Condition と Post Conditionの一致を確かめる部分に着手中

図解


Meta Data Gear
Data Gear
Meta
Code Gear
Code Gear
Meta
Code Gear
Meta Data Gear
Data Gear
Pre Condition
Data Gear
Pre
Condition
Proof
Code Gear
Post
Condition
Proof
Post Condition
Data Gear

課題

- Tree の Node が正しい大小関係を持っているのか検証する方法
  - Fresh List という順序関係を持った List を使用した方が望ましい
    - Fresh List への insert が複雑で実装が困難になった
    - top との順序関係を持つことで大小 sort された Stack で代用している

- 健全性の証明の仕方が不明
  - コマンド一つ一つの検証をした後、それが次のコマンドに正常に接続されているか検証する必要がある。

これから

- 大学院ではCbCのコードから検証用のAgdaコードの自動化または半自動化について研究予定
  - これを用いると、CbCで書かれたコード全ての信頼性を検証できる