社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

GearsOS の Hoare Logic をベースにした検証手法

外間　政尊†　　河野　真治††

† 琉球大学大学院理工学研究科情報工学専攻
†† 琉球大学工学部情報工学科
E-mail: †{masataka,kono}@cr.ie.u-ryukyu.ac.jp

あらまし　あらまし
キーワード　プログラミング言語, CbC, Gears OS, Agda

Masataka HOKAMA† and Shinji KONO††

† Interdisciplinary Information Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†† Information Engineering, University of the Ryukyus.
E-mail: †{masataka,kono}@cr.ie.u-ryukyu.ac.jp

1. ま え が き

Gears OS は継続を主とするプログラミング言語 CbC で記述されている。OS やアプリケーションの信頼性を上げるには仕様を満たしていることを確認する必要がある。現在 GearsOS の仕様の確認には定理証明系である Agda を用いている。CbC では関数呼び出しを用いず goto 文により遷移する。これは Agda 上では継続渡しの記述を用いた関数として記述する。また、継続にはある関数を実行するための事前条件や事後条件などをもたせることが可能である。Floyd–Hoare logic (以下 Hoare Logic) は事前条件が成り立っているときにある関数を実行して、それが停止する際に事後条件を満たすことを確認することで、検証を行うアイディアである。これは継続を用いた Agda 上では事前条件とその証明をを継続で関数に渡し、関数からさらに継続した先で事後条件とその証明が成り立つという形で記述できる。本研究では GearsOS の仕様確認に Hoare Logic をベースとした証明を導入し、今まで行っていた証明との比較を行う。

2. GearsOS

Gears OS は信頼性をノーマルレベルの計算に対して保証し、拡張性をメタレベルの計算で実現することを目標に開発している OS である。

Gears OS は処理の単位を Code Gear、データの単位を Data Gear と呼ばれる単位でプログラムを構成する。信頼性や拡張性はメタ計算として、通常の計算とは区別して記述する。

3. CodeGear と DataGear

Gears OS ではプログラムとデータの単位として CodeGear、DataGear を用いる。 Gear は並列実行の単位、データ分割、Gear 間の接続等になる。CodeGear はプログラムの処理そのもので、図 1 で示しているように任意の数の Input DataGear を参照し、処理が完了すると任意の数の Output DataGear に書き込む。

CodeGear 間の移動は継続を用いて行われる。継続は関数呼び出しとは異なり、呼び出した後に元のコードに戻らず、次の CodeGear へ継続を行う。これは、関数型プログラミングでは末尾関数呼び出しを行うことに相当する。

Gear では処理やデータ構造が CodeGear、 DataGear に閉じている。したがって、DataGear は Agda のデータ構造 (data と record) で表現できる。CodeGear は Agda の CPS (Continuation Passing Style) で関数として表現することができる。

また Gears OS 自体もこの Code Gear、Data Gear を用いた CbC(Continuation based C) で実装される。そのため、Gears OS の実装は Code Gear、Data Gear を用いたプログラミングスタイルの指標となる。

4. Agda

5. CbC と Agda

ここでは CodeGear、DataGear を用いたプログラムを検証するため、Agda 上での CodeGear、 DataGear の対応をみて

いく。

CodeGear は Agda では継続渡しで書かれた関数と等価である。継続は不定の型 (t) を返す関数で表される。CodeGear 自体も同じ型 t を返す関数となる。例えば、 Stack への push を行う関数 pushStack は以下のような型を持つ。

Code 1: pushStack の型

```
1  pushStack : a -> (Stack a si -> t) -> t
```

pushStack が関数名で、コロンの後ろに型を記述する。最初の引数は Stack に格納される型 a を持つ。二つ目の引数は継続であり、Stack a si (si という実装を持つ a を格納する Stack) を受け取り不定の型 t を返す関数である。この CodeGear 自体は不定の型 t を返す。

GearsOS で CodeGear の性質を証明するには、 Agda で記述された CodeGear と DataGear に対してメタ計算として証明を行う。証明すべき性質は、不定の型を持つ継続 t に記述することができる。例えば、 Stack にある値 x を push して、pop すると x' が取れてくる。Just x と Just x' は等しい必要がある。これは Agda では (Just x ≡ x') と記述される。ここで Just とは Agda の以下のデータ構造である。

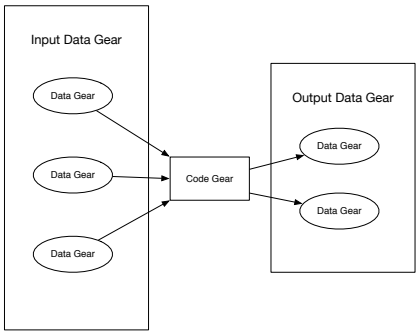

図 1: CodeGear と DataGear の関係

Code 2: data 型の例:Maybe

```
1  data Maybe {n : Level } (a : Set n) : Set n where
2    Nothing : Maybe a
3    Just    : a -> Maybe a
```

これは DataGear に相当し、Nothing と Just の二つの状態を保つ。pop した時に、 Stack が空であれば Nothing を返し、そうでなければ Just のついた返り値を返す。

この性質を Agda で表すと、以下のような型になる。Agda では証明すべき論理式は型で表される。継続部分に直接証明すべき性質を型として記述できる。Agda ではこの型に対応する λ 項を与えると証明が完了したことになる。

Code 3: push と pop

```
1  push->pop : {l : Level} {D : Set l} (x : D )
2          (s : SingleLinkedStack D) ->
3          pushStack (stackInSomeState s)
4          x (\s -> popStack s
5          ( \s3 x1 -> (Just x ≡ x1)))
```

このように、CodeGear を Agda で記述し、継続部分に証明すべき性質を Agda で記述する。

6. HoareLogic

HoareLogic とは C.A.R Hoare、 R.W Floyd らによるプログラムの検証のアイディアである。これはプログラムの部分的な正当性を検証するアイディアで、事前条件 (Pre-Condition) P が成り立つとき、コマンド C を実行した後に事後条件 (Post-Condition) Q が成り立つ。これを *PCQ* のように表し、コマンドをつなげてプログラム全体を記述することで、プログラム内のすべての部分について検証を行うことができる。

HoareLogic ではプログラムを Assign、 Sequential Composition、 If、 While というコマンドで記述する。

7. Agda での HoareLogic

本章では、Agda 上での HoareLogic についての記述と検証を行う。今回は、4 のような while Loop に対しての検証を行うこととする。

Code 4: while Loop

```
n = 10;
i = 0;

while (n>0)
{
  i++;
  n--;
}
```

4 では最初期の事前条件は何もなく、プログラムが停止するときの条件として、 $i == 10 \wedge n == 0$ が成り立つ。また、$n = 10$、$i = 0$、 といった代入に事前条件と、事後条件をつけ、while 文 にループの普遍条件をつけることになる。

5 は Agda 上での HoareLogic の構築子である。

Env は 4 の n、 i といった変数をまとめたものであり、型として Agda 上での自然数の型である Nat を持つ。PrimComm は Primitive Command で、 n、i といった変数に 代入するときに使用される関数である。Cond は Condition で、 変数を受け取って　Bool 値を返す関数となっている。Agda のデータで定義されている Comm は HoareLogic での Command を表す。Skip は何も変更しない Command で、 Abort はプログラムを中断する Command である。

PComm は PrimComm を受けて Command を返す型で定義されていて、変数を代入するときに使われる。Seq は Sequence Composition で Command を 2 つ受けて Command を返す型で定義されている。これは、ある Command から Command に移り、その結果を次の Command に渡す型になっている。If は Cond と Comm を 2 つ受け取り、 Cond の中身によって Comm を変える Command である。While は Cond と Comm を受け取り、 Cond の中身が True である間、 Comm を繰り返す Command である。

Code 5: Agda での HoareLogic の構成

```
record Env : Set where
  field
    varn : ^^e2^^84^^95
    vari : ^^e2^^84^^95
open Env

PrimComm : Set
PrimComm = Env → Env

Cond : Set
Cond = (Env → Bool)

data Comm : Set where
  Skip  : Comm
  Abort : Comm
  PComm : PrimComm -> Comm
  Seq   : Comm -> Comm -> Comm
  If    : Cond -> Comm -> Comm -> Comm
  While : Cond -> Comm -> Comm
```

Agda 上の HoareLogic で使われるプログラムは Comm 型の関数となる。プログラムの処理を Seq でつないでいき、最終的な状態にたどり着くと値を返して止まる。6 は 4 で書いた While Loop を HoareLogic での Comm で記述したものである。ここでの $ は () の対応を合わせる Agda の糖衣で、行頭から行末までを () で囲っていることと同義である。

Code 6: HoareLogic のプログラム

```
program : Comm
program =
    Seq ( PComm λ ( env → record env {varn = 10}))
    $ Seq ( PComm λ ( env →
     record env {vari = 0}))
    $ While λ ( env → lt zero (varn env ) )
      (Seq (PComm λ ( env →
     record env {vari = ((vari env) + 1)} ))
        $ PComm λ ( env →
     record env {varn = ((varn env) - 1)} ))
```

しかし、 この Comm は Command をならべているだけである。そのため、この Command を Agda 上で実行するため、7 のような interpreter を記述した。

Code 7: Agda での HoareLogic interpreter

```
{-# TERMINATING #-}
interpret : Env → Comm → Env
interpret env Skip = env
interpret env Abort = env
interpret env (PComm x) = x env
interpret env (Seq comm comm1) = interpret (
    interpret env comm) comm1
interpret env (If x then else) with x env
... | true = interpret env then
... | false = interpret env else
interpret env (While x comm) with x env
... | true = interpret (interpret env comm) (While
    x comm)
... | false = env
```

7 は 初期状態の Env と 実行する Command の並びを受けとって、実行後の Env を返すものとなっている。

Code 8: Agda での HoareLogic の実行

```
test : Env
test =  interpret ( record { vari = 0  ; varn = 0 }
     ) programt
```

8 のように interpret に $vari = 0, varn = 0$ の record を渡し、 実行する Comm を渡して 評価してやると $recordvarn = 0; vari = 10$ のような Env が返ってくる。

次に先程書いたプログラムの証明について記述する。

9 は Agda 上での HoareLogic での証明の構成である。HTProof では Condition と Command もう一つ Condition を受け取って、Set を返す Agda のデータである。– これは Pre と Post の Condition を Command で変化させる

Code 9: Agda での HoareLogic の構成

```
data HTProof : Cond -> Comm -> Cond -> Set where
  PrimRule : {bPre : Cond} -> {pcm : PrimComm} -> {
    bPost : Cond} ->
             (pr : Axiom bPre pcm bPost) ->
             HTProof bPre (PComm pcm) bPost
  SkipRule : (b : Cond) -> HTProof b Skip b
  AbortRule : (bPre : Cond) -> (bPost : Cond) ->
              HTProof bPre Abort bPost
  WeakeningRule : {bPre : Cond} -> {bPre' : Cond}
    -> {cm : Comm} ->
                {bPost' : Cond} -> {bPost : Cond}
    ->
                Tautology bPre bPre' ->
                HTProof bPre' cm bPost' ->
                Tautology bPost' bPost ->
                HTProof bPre cm bPost
  SeqRule : {bPre : Cond} -> {cm1 : Comm} -> {bMid
    : Cond} ->
            {cm2 : Comm} -> {bPost : Cond} ->
            HTProof bPre cm1 bMid ->
            HTProof bMid cm2 bPost ->
            HTProof bPre (Seq cm1 cm2) bPost
  IfRule : {cmThen : Comm} -> {cmElse : Comm} ->
           {bPre : Cond} -> {bPost : Cond} ->
           {b : Cond} ->
           HTProof (bPre /\ b) cmThen bPost ->
           HTProof (bPre /\ neg b) cmElse bPost ->
           HTProof bPre (If b cmThen cmElse) bPost
  WhileRule : {cm : Comm} -> {bInv : Cond} -> {b :
    Cond} ->
              HTProof (bInv /\ b) cm bInv ->
              HTProof bInv (While b cm) (bInv /\
    neg b)
```

9 を使って先程の while Program を証明する。証明は 9 の proof1 の様になる。

Code 10: Agda 上での WhileLoop の検証

```
initCond : Cond
initCond env = true

stmt1Cond : Cond
stmt1Cond env = Equal (varn env) 10

stmt2Cond : Cond
stmt2Cond env = (Equal (varn env) 10) ∧
     (Equal (vari env) 0)

whileInv : Cond
whileInv env = Equal ((varn env) + (vari env)) 10

whileInv' : Cond
whileInv' env = Equal ((varn env) + (vari env)) 11

termCond : Cond
termCond env = Equal (vari env) 10

proof1 : HTProof initCond program termCond
proof1 =
      SeqRule λ{ e →
     true} ( PrimRule empty-case )
    $ SeqRule λ{ e →
       Equal (varn e) 10} ( PrimRule lemma1   )
    $ WeakeningRule λ{ e → (Equal (varn e) 10) ∧
     (Equal (vari e) 0)}  lemma2 (
           WhileRule {_} λ{ e →
     Equal ((varn e) + (vari e)) 10}
           $ SeqRule (PrimRule λ{ e →
      whileInv e  ∧  lt zero (varn e) } lemma3 )
                   $ PrimRule {whileInv'} {_} {
    whileInv}  lemma4 ) lemma5
```

proof1 は 6 の program と似た形をとっている。Comannd に対応する証明規則があるため、証明はプログラムに対応する形になる。

Code 11: Gears 上での WhileLoop の検証

```
proofGears : {c10 :  ^^e2^^84^^95 } → Set
proofGears {c10} = whileTest' {_} {_} {c10} λ
     ( n p1 →  conversion1 n p1  λ ( n1 p2 →
      whileLoop' n1 p2 λ ( n2 →  ( vari n2 ≡
      c10 ))))

proofGearsMeta : {c10 :  ^^e2^^84^^95 } →
      whileTest' {_} {_} {c10} λ ( n p1 →
       conversion1 n p1λ ( n1 p2 →
      whileLoop' n1 p2 λ ( n2 →  ( vari n2 ≡
      c10 ))))
proofGearsMeta {c10} = {!!}
```

文　献

[1] 外間政尊, 河野真治, GearsOS の Agda による記述と検証, システムソフトウェアとオペレーティング・システム研究会, 2018.