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 triple 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。Hoare triple で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。これは継続を用いた Agda 上では事前条件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が成り立つという形で記述できる。本発表では GearsOS の仕様確認に Hoare triple 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。

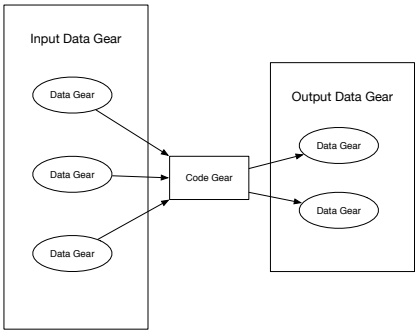

図 1: CodeGear と DataGear の関係

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。

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。

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。

GearsOS での記述は interface によってモジュール化される。よって、このモジュール化も Agda により記述する必要がある。CbC で記述された任意の CodeGear と Meta CodeGear が Agda にそのまま変換されるわけではないが、変換可能なように記述されると仮定する。

4. Agda と GearsOS

Agda と GearsOS

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.