継続を用いたxv6 kernelの書き換え

坂本昂弘[†1]　桃 原 優[†1]　河野真治[†1]

概要：xv6 は MIT で教育用の目的で開発されたオペレーティングシステムで基本的な Unix の構造を持っている。当研究室で開発している Continuation based C (CbC) は継続を中心とした言語で、用いることにより検証しやすくなる。xv6 を CbC で書き換えることで、 オペレーティングシステムの基本的な機能に対する検証を行うために書き換えを行う。

1. OS の拡張性と信頼性の両立

OS はさまざまなコンピュータの信頼性の基本である。OS の信頼性を保証する事自体が難しいが、時代とともに進歩するハードウェア、サービスに対応して OS 自体が拡張される必要があり、さらに非決定的な実行を持つ為、モデル検査は無限の状態ではなくても巨大な状態を調べることになり、状態を有限に制限したり状態を抽象化したりする方法が用いられている。証明やモデル検査を用いて OS を検証する方法はさまざまなものが検討されている。検証は一度ですむものではなく、アプリケーションやサービス、デバイスが新しくなること に検証をやり直す必要がある。つまり信頼性と拡張性を両立させることが重要である。コンピュータの計算はプログラミング言語で記述されるが、その言語の実行は操作的意 味論の定義などで規定される。プログラミング言語で記述される部分をノーマルレベル の計算と呼ぶ。実際にコードが実行される時には、処理系の詳細や使用する資源、あるい は、コードの仕様や型などの言語以外の部分が服属する。これをメタレベルの計算とい う。この二つのレベルを同じ言語で記述できるようにして、ノーマルレベルの計算の信頼 性をメタレベルから保証できるようにしたい。ノーマルレベルでの正当性を保証しつつ、 新しく付け加えられたデバイスやプログラムを含む正当性を検証したい。ノーマルレベルとメタレベルを共通して表現できる言語として Continuation based C(以 下 CbC) を用いる。CbC は関数呼出時の暗黙の環境 (Environment) を使わずに、コー ドの単位を行き来できる引数付き goto 文 (parametarized goto) を持つ C と互換性のある 言語である。これを用いて、Code Gear と Data Gear、さらにそのメタ構造を導入する。 これらを用いて、検証された Gears OS を構築したい。Meta Gear を入れかえることにより、ノーマルレベルの Gear をモデル検査することが できるようにしたい。ノーマルレベルでの Code Gear と Data Gear は継続を基本とした 関数型プログラミング的な記述に対応する。この記述を定理証明支援系である Agda を用 いて直接に証明できるようにしたい。Gears OS の記述はそのまま Agda に落ちる。Code Gear、Data Gear を用いた言語である CbC で記述することによって検証された OS を実 装することができる。

2. Continuation based C

2.1 CbC の概要

CbC は処理を Code Gear とした単位を用いて記述するプログラミング言語である。 Code Gear 間では軽量継続 (goto 文) による遷移を行うので、継続前の Code Gear に戻 ることはなく、状態遷移ベースのプログラミングに適している。図 1 は Code Gear 間 の処理の流れを表している。

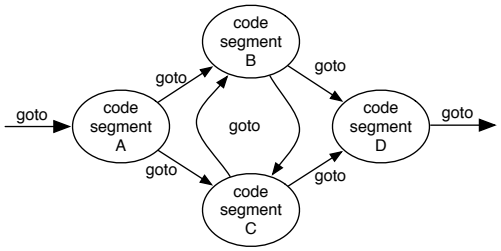

図 1: goto による code gear 間の継続

現在 CbC は C コンパイラである GCC 及び LLVM をバックエンドとした clang、MicroC コンパイラ上で実装さ

[†1] 現在，琉球大学工学部情報工学科
Presently with Information Engineering, University of the Ryukyus.
[†2] 現在，琉球大学大学院理工学研究科情報工学専攻
Presently with Interdisciplinary Information Engineering, Graduate School of Engineering and Science, University of the Ryukyus.

れている。今回 xv6 の kernel の部分をこの CbC を用いて書き換える。

2.2 Code Gear

Code Gear は CbC における最も基本的な処理単位である。ソースコード 1 は CbC における Code Gear の一例である。

ソースコード 1: code segment の軽量継続

```
1 __code cs0(int a, int b){
2   goto cs1(a+b);
3 }
4 __code cs1(int c){
5   goto cs2(c);
6 }
```

Code Gear は__code Code Gear 名 (引数) の形で記述される。Code Gear は戻り値を持 たないので、関数とは異なり return 文は存在しない。次の Code Gear への遷移は goto Code Gear 名 (引数) で次の Code Gear への遷移を記述する。ソースコード 1 での goto cs1(a+b); がこれにあたる。この goto の行き先を継続と呼び、このときの a+b が次の Code Gear への出力となる。Scheme の継続と異なり CbC には呼び出し元の環境がない ので、この継続は単なる行き先である。したがってこれを軽量継続と呼ぶこともある。cs1 へ継続した後は cs0 へ戻ることはない。軽量継続により、並列化、ループ制御、関数コー ルとスタックの操作を意識した最適化がソースコードレベルで行えるようにする。CbC は軽量継続による遷移を行うので、継続前の Code Gear に戻ることはなく、状態 遷移ベースのプログラミングに適している。

2.3 Data Gear

Data Gear は CbC におけるデータの単位である。CbC では Code Gear は Input Data Gear、Output Data Gear を引数に持ち、任意の Input Data Gear を参照し、Output Data Gear を書き出す。図 2 Code Gear はこのとき渡された引数の Data Gear 以外を参照することはない。この Data Gear の対応から依存関係の解決を行う。

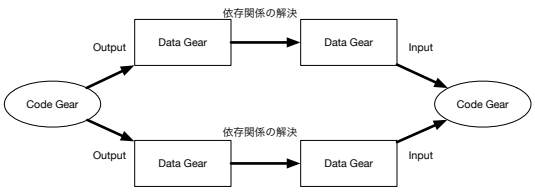

図 2: CodeGear と DataGear

3. Gears OS の概要

Gears OS は Code Gear、Data Gear の単位を用いて開発されており、CbC で記述されて いる。Gears OS は Context と呼ばれる使用されるすべての Code Gear、Data Gear 持っ ている Meta Data Gear を持つ。Gears OS は継続の際この Context を常に持ち歩き、必 要な Code Gear、Data Gear を参照したい場合、この Context を通して参照する。

4. xv6 の CbC への書き換え

4.1 現状

4.2 書き換え

5. 結論

5.1 評価

5.2 今後

5.3

参考文献

[1] 宮城光希: **継続を基本とした言語による OS のモジュール化**. 琉球大学理工学研究科修士論文、2019