Agda と継続を用いたプログラムの検証
Verification of program using Agda and continuation

145750B 氏名 外間政尊 指導教員：河野 真治

Abstract

Program has high reliability. it is important. we are proposing to write program in units of CodeGear, DataGear for increase the reliability. we are deceloping Continuation based C (CbC) that can use units CodeGear and DataGear. In CbC, the handling of data in existing implementations is complicated. for that purpose, we can provide a interface mechanisms which are packages of CodeGears and DataGears. we made these units and interface available for Agda. Also converted Stack and Tree wrote in CbC to Agda. In this papaer, we tried several proofs on them.

1 ソフトウェアの信頼性の保証

ソフトウェアの信頼性を保証することは重要である。現在ソフトウェアの信頼性を保証する方法として代表的なものはモデル検査と、定理証明が存在している。当研究室では検証しやすいプログラムの単位として、CodeGear と DataGear という単位を用いるプログラミングスタイルを提案している。また、 CodeGear 、 DataGear という単位を用いてプログラミングする言語として Countinuation based C (以下 CbC) を開発している。

CbC では自由に DataGear を扱う際に直接 CodeGear からアクセスできてしまうと信頼性を損ねる。その為に stub CodeGear と呼ばれるデータを扱うための Meta CodeGear を定義している。基本的な stub CodeGear は CodeGear から自動で生成することができるようになっている。しかし、CbC で実装していくにつれて stub CodeGear の記述が複雑になることが分かった。 その為、既存の実装をモジュールとして扱うために Interface という仕組みを導入した。

本研究では CbC 、 Agda の両方で Interface を実装し、Interface を含めた Stack と Tree の性質を部分的に証明した。

2 Countinuation based C (CbC)

Continuation based C (CbC) とは、当研究室で開発されているプログラミング言語である。CbC は C 言語とほぼおなじ構文を持つが、 C の関数の代わりに CodeGear を用いて処理を記述する。CodeGear は処理の単位でそれらの状態を goto で遷移して記述する。この goto による処理の遷移を継続と呼ぶ。DataGear は CodeGear が扱うデータの単位であり、処理に必要なデータが全て入っている。次の CodeGear に処理を移す際は、 goto の後に CodeGear 名と DataGear を指定する。CbC ではこの継続処理がメタ計算として定義されており、 CodeGear に変更なく検証等を行うことができる。例として CbC による Stack に対する操作のコード示す。

ソースコード 1: CbC による Stack

```
__code pushSingleLinkedStack(struct Single LinkedStack*
      stack,union Data* data, __code next(...)) {
    Element* element = new Element();
    element->next = stack->top;
    element->data = data;
    stack->top = element;
    goto next(...);
}
__code popSingleLinkedStack(struct Single LinkedStack*
     stack, __code next(union Data* data, ...)) {
    if (stack->top) {
        data = stack->top->data;
        stack->top = stack->top->next;
    } else {
        data = NULL;
    }
    goto next(data, ...);
}
```

push では新しい element を作成し next に現在の top を入れ、Stack の top を書き換えて次の CodeGear に飛ぶ。pop では top に data があればそれを next に入れ、 Stack を更新して次の CodeGear に飛ぶ。 top に data が無ければ NULL を next に入れて次の CodeGear に飛ぶ。

3 CbC における Interface の実装

CbC で実装していくにつれ、stub CodeGear の記述が複雑になった。そのため 既存の実装を モジュールとして扱うため Interface を導入した。Interface は DataGear に対して何らかの操作 (API) を行う CodeGear とその CodeGear で使われる DataGear の集合を抽象化した メタレベルの DataGear として定義した。例として Stack での Interface の実装をみる。

ソースコード 2: CbC での Stack-Interface の実装

```
Stack* createSingleLinkedStack(struct Context* context)
    {
```

```
    struct Stack* stack = new Stack();
    struct SingleLinkedStack* singleLinkedStack = new
      SingleLinkedStack();
    stack->stack = (union Data*)singleLinkedStack;
    singleLinkedStack->top = NULL;
    stack->push = C_pushSingleLinkedStack;
    stack->pop  = C_popSingleLinkedStack;
    /* 途中省略 */
    return stack;
}
```

元の実装の push では Stack を指定する必要があるが、Interface での実装は push 先の Stack が stackImpl として扱われている。この stackImpl は呼ばれた時の Stack と同じになる。これにより、 ユーザーは実行時に Stack を指定する必要がなくなる。このように Interface 記述をすることで CbC で通常記述する必要がある一定の部分を省略し呼び出しが容易になる。

4 Agda における Interface の実装

Agda でも CbC と同様に Interface の実装した。例として Agda で実装した Stack 上の interface をみる。Stack の実装は SingleLinkedStack(ソースコード 3) として書かれている。それを Stack 側から interface を通して呼び出している。

ソースコード 3: Agda における Stack の実装

```
-- Implementation
pushSingleLinkedStack : {n m : Level } {t : Set m } {
     Data : Set n} -> SingleLinkedStack Data -> Data
      -> (Code : SingleLinkedStack Data -> t) -> t
pushSingleLinkedStack stack datum next = next stack1
  where
    element = cons datum (top stack)
    stack1  = record {top = Just element}


popSingleLinkedStack : {n m : Level } {t : Set m } {a :
     Set n} -> SingleLinkedStack a -> (Code :
     SingleLinkedStack a -> (Maybe a) -> t) -> t
popSingleLinkedStack stack cs with (top stack)
...                          | Nothing = cs stack  Nothing
...                          | Just d  = cs stack1 (Just
     data1)
  where
    data1  = datum d
    stack1 = record { top = (next d) }
```

5 Agda による Interface 部分を含めた Stack の部分的な証明

Agda で Interface を定義する事ができた。この Interface を通した Stack で push 、 pop などの操作が正しく行われるかの証明を行った。ここでの証明とは Stack の処理が特定の性質を持つことを保証することである。

Stack の処理として様々な性質が存在するが、ここでは「どのような状態の Stack でも、値を push した後 pop した値は直前に入れた値と一致する」という性質を証明した。

まず始めに不定状態の Stack をソースコード 4 で定義した。 stackInSomeState が不定状態の Stack である。ソースコード 4 の証明ではこの stackInSomeState に対して、 push を 2 回行い、 pop2 をして取れたデータは push したデータと同じものになることの証明している。

ソースコード 4: 抽象的な Stack の定義と push− >push− >pop2 の証明

```
stackInSomeState : {l m : Level } {D : Set l} {t : Set m
     } (s : SingleLinkedStack D ) -> Stack {l} {m} D {t
     } ( SingleLinkedStack  D )
stackInSomeState s =  record { stack = s ; stackMethods
     = singleLinkedStackSpec }

push->push->pop2 : {l : Level } {D : Set l} (x y : D )
     (s : SingleLinkedStack D ) ->
pushStack ( stackInSomeState s )  x ( \s1 -> pushStack
     s1 y ( \s2 -> pop2Stack s2 ( \s3 y1 x1 -> (Just x
     ≡  x1 ) ∧  (Just y ≡  y1 ) ) ))
push->push->pop2 {l} {D} x y s = record { pi1 = refl
     ; pi2 = refl }
```

stackInSomeState 型の s が抽象的な Stack で、そこに x 、 y の 2 つのデータを push している。また、 pop2 で取れたデータは y1 、 x1 となっていて両方が Just で返ってくるかつ、 x と x1 、 y と y1 がそれぞれ合同であることが仮定として型に書いた。関数本体で返る値は $x \equiv x1$ と $y \equiv y1$ で両方共に成り立つ為、refl で推論が通る。これにより、抽象化した Stack に対して push を 2 回行い、 pop を行うと push したものと同じものを受け取れることが証明できた。

6 まとめ

本研究では CodeGear、 DataGear を用いたプログラミング手法を用いて、Agda で Interface を用いたプログラムを実装、検証した。また、 CbC で記述した時には細かく分かっていなかった Interface の型が明確になった。今後の課題としては、CodeGear、DataGear をベースにした Hoare Logic を Agda で実装する。また、その Hoare Logic を使い、いくつかの証明を実際に記述する。

参考文献

[1] 比嘉 健太, 河野 真治. メタ計算を用いた Continuation based C の検証手法, 2016.

[2] 伊波 立樹, 東恩納 琢偉, 河野 真治. Code Gear 、Data Gear に基づく OS のプロトタイプ, 2016.

[3] 徳森 海斗, 河野 真治. LLVM Clang 上の Continuation based C コンパイラ の改良, 2015.

[4] The agda wiki. http://wiki.portal.chalmers.se/agda/pmwiki.php. Accessed: 2017/10/24(Tue).

[5] Welcome to agda ' s documentation! — agda 2.6.0 documentation. http://agda.readthedocs.io/en/latest/index.html. Accessed: 2017/10/24(Tue).