神奈川大学理学部　情報科学科　馬谷研究室

レジスタマシンをターゲットとするコンパイラの依存型言語 Agda を用いた実装

秋田 一輝 201902949

1　はじめに

レジスタマシンをターゲットとする「正しいコンパイラ」を依存型言語 Agda によって実装する．具体的には，ソース言語，マシン語コード，ソース言語からマシン語コードへのコンパイラ，マシン語コードを実行するレジスタマシンをそれぞれ Agda で記述する．Agda を用いることによりマシン語コードの型の中に実行前後のマシンの状態を表現でき，おかしな動作記述になっていないことが型検査により保証される．スタックマシンをターゲットとするコンパイラの Agda による実装については先行研究 [3] があり，本研究では，それを基にレジスタマシンに特有な部分の適切な型表現を考案した上でレジスタマシンを定義している．

2　スタックマシンの Agda による実装

スタックマシンをターゲットとするコンパイラの Agda を用いた実装として，Haskell を用いた先行研究 [1] に基づく先行研究 [3] がある．実装は，ソース言語（式 Exp），評価関数 eval，マシンの状態を表すスタック Stack，マシン語コード Code，コンパイル関数 comp，マシン語コード実行関数 exec からなる．

```
data Code : List Set → List Set → Set₁
  where
    PUSH : T → Code (T :: S) S' → Code S S'
    ...
comp : Exp T → Code (T :: S) S' → Code S S'
exec : Code S S' → Stack S → Stack S'
```

ここでは，実行時スタックの型情報 S（各要素の型からなるリスト）を Agda の List Set 型の値により表し，実行時スタックの（Agda の）型は S を引数とする依存型 Stack S で表す．また，マシン語コード Code の型は実行前後の実行時スタックの型情報を含んでいる．実行時スタックの各要素の型のリストの変化を型の中に埋め込むことで，マシン語コード Code が意図通りの動作をすることを保証できる．

例えば，PUSH 命令では，まず，第 1 引数に受け取った T 型の値を積んだスタックを，実行開始時の状態とするマシン語コード Code を継続として第 2 引数に受け取る．そして，PUSH 命令の実行開始時のスタックに第 1 引数で受け取った T を積み，その後，第 2 引数で受け取った継続を実行する．このように，PUSH 命令は引数をスタックに積むことを型により表している．

また，コンパイル関数 comp は引数に受け取る式と継続から，それに基づいたマシン語コード Code を返す．その際，そのマシン語コードと継続は実行前後の状態の型情報が入ったマシン語コード Code になっていることが型によって保証されている．

実行関数 exec は引数に受け取ったマシン語コード Code と実行前の状態の型情報 S から，実行後の状態の型情報 S' を返す．これもまた，適切に動作することが型によって保証されている．

これらの定義に基づき，以下の等式は，定義されている comp が「正しいコンパイラ」であることを示している．ここで，e は Exp，c はマシン語コード，s はスタック，▷ はスタックへのプッシュを意味する．

```
exec (comp e c) s = exec c (eval e ▷ s)
```

3　レジスタマシンの Haskell による実装

レジスタマシンをターゲットとするコンパイラの Haskell を用いた実装として先行研究 [2] がある．実装はスタックマシンとほぼ同様であるが，以下が異なる．

```
type Conf = (Acc,Mem)
comp :: Expr → Reg → Code → Code
exec :: Code → Conf → Conf
```

レジスタマシンの状態は Conf で表される．レジスタ番号 Reg は整数であり，レジスタ番号から値への連想リス

トをメモリ Mem としている．また，アキュムレータ Acc は一時的な記憶領域としての特別なレジスタである．

コンパイル関数 comp が第 2 引数に受け取るレジスタ番号は，それ以降のレジスタが空いていることを表すものである．2 節の Agda の Code とは異なり，上記の Haskell の Code 型の中にはマシンの実行状態が埋め込まれておらず，コード実行前後における状態の変化の正しさを保証できていない．同様の問題は，実行関数 exec においても生じている．

4 レジスタマシンの Agda による実装

本研究ではレジスタマシンをターゲットとするコンパイラの Agda を用いた実装を行う．3 節の先行研究 [2] と異なる点として，以下の 2 点が挙げられる．

まず，Value 型はマシンが扱う値の型である．

```
data Value : Set where
  v-value : Value
  n-value : N → Value
  b-value : Bool → Value
```

コンストラクタには，値がないこと（void）を表す v-value，自然数を表す n-value，ブール値を表す b-value がある．Value 型の値は，それが表す値だけでなく，型の情報も含んでいることに注意してほしい．

次に，マシン語コード Code の型は実行前後の状態 Conf の変化を依存型を使って詳細に記述するようにする．

```
data Code : Conf → Conf → Set
  LOAD : ∀{v : Value} {m : Mem}
           {c : Conf} (w : Value)
         → Code (w , m) c
         → Code (v , m) c
  STORE : ∀ {v : Value} {m : Mem}
            {c : Conf} (r : Reg)
          → Code (v , (r , v) :: m) c
          → Code (v , m) c
  ADD : ∀ {n n' : N} {m : Mem}
          {c : Conf} (r : Reg)
        → Code (n-value (n + n') ,
                (r , n-value n) :: m) c
        → Code (n-value n' ,
                (r , n-value n) :: m) c
  HALT : ∀ {c : Conf} → Code c c
```

例えば，STORE 命令では，実行することにより状態 (v , m) が状態 (v , (r , v) :: m) へ変化することを，コンストラクタの型情報として埋め込んでいる．

特に，Value 型の変数 v が値も含んでいることに注意してほしい．そのため，上記の Code 型の定義自体がマシン語コードの実行を完全に記述しており，これとは別に実行関数 exec を定義する必要がなくなっている．

5 まとめ

レジスタマシンをターゲットとするコンパイラの Agda を用いた実装を，先行研究 [1][2][3] を参考にしつつ構築した．Agda を用いることにより，マシン語コードの型へ実行前後のマシン状態の変化を埋め込んだ結果，マシン語コードの型の定義自体が実行関数の役割も担っている．ただし，本研究ではコンパイル関数の定義はまだ完成していない．4 節の定義を用いて，マシンの状態の変化を詳細に記述した型を持つ Code 型に基づくコンパイル関数 comp を定義することは今後の課題の一つである．

謝辞

本研究を進めるにあたり，神奈川大学 木下佳樹教授に多大なご助言を頂きました．心より感謝申し上げます．

参考文献

[1] Patric Bahr, Graham Hutton, *Calculating correct compilers*, Jounal of Funtional Programming, 25, 2015

[2] Patric Bahr, Graham Hutton, *Calculating Correct Compilers II: Return of the Register Machines.*, Jounal of Funtional Programming, 30, 2020

[3] Mitchell Pickard, Graham Hutton, *Calculating Dependently-Typed Compilers (functional pearl)*, Proceedings of the ACM on Programming Languages, Vol. 5, No. ICFP, 2021