神奈川大学理学部　情報科学科　馬谷研究室

Featherweight Java の型検査の turnstile 上での実装

菅原 裕太 201703407

1 はじめに

静的型付き言語ではどのプログラムにも型というものが存在する．Int や Bool のような型を使用する際には型のエラーなく実行されるかどうかを知る「型検査」という処理が必要となってくる．この型検査について Racket 言語上に静的型付き言語を実装するためのライブラリ turnstile[1] が存在する．turnstile ではこれまでに数多くの型付き言語が作られているが，オブジェクト指向言語を実装しているものはほとんど存在しない．簡易なオブジェクト指向言語のサブセットを設定し，その型システムの実装を turnstile 上で行えば，より本格的なオブジェクト指向言語の実装にも役立つのではないかと考えられる．本研究では Featherweight Java (FJ) というオブジェクト指向言語における型検査部分を実装する．まず本研究では turnstile と FJ の二つの言語の理解が必要となってくるため第 2 節では turnstile についての説明，第 3 節では FJ についての説明を行う．第 4 節では実際に本研究で実装した型検査プログラムを簡潔に説明し最後にまとめを行う．

2 turnstile

```
(define-typed-syntax
    λ #:datum-literals (:)
     [(_ ([x:id : τ_in:type] ...) e)
      ≫ [[x ≫ x- : τ_in.norm]
        ...⊢ e  ≫ e- ⇒ τ_out]
    ---------
    [⊢ (λ- (x- ...) e-)]])
```

このコードは turnstile におけるラムダ式を検査するコードの一部である．まず初めに `define-typed-syntax` によりその言語の定義を行う．3 行目では環境 `e` のなかで `t_in` という名前の型を持つ `x` という引数が 0 個以上存在していることを表している．5 行目では `e` が展開され `e-` となった時に `t_out` という型を持っている．という検査が行われている．このような処理が 0 個以上続いているため一つずつ評価を行っている．turnstile ではこのように型環境を用いながら，それに伴った型や引数を展開しながら型検査を行う．

3 Featherweight Java

FJ[2] は Java の型システムをモデル化する最小限の核となる計算体系の候補である．FJ はコンパクトさを重視したものであるため項の形式にはオブジェクト生成，メソッド呼び出し，フィールド参照，キャスト，変数の 5 つしか存在していない．この FJ のプログラムはクラス定義の集まりと評価される項からなる．以下に FJ の典型的なクラス定義を示す．

```
class A extends Object { A() { super(); }}
class B extends Object { B() { super(); }}
class Pair extends Object {
  Object fst;
  Object snd;
  // Constructor:
  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd;}
  // Method definition:
  Pair setfst(Object newfst){
    return new Pair(newfst, this.snd); }}
```

クラス定義では初めにクラス宣言が行われる．文法

を規則的にするためにスーパークラス，コンストラクタなども常に書く必要がある．このコードにおいて項の形式は次のようになる．

- オブジェクト生成：new A(),new B()
- メソッド呼び出し：setfst(...)
- フィールド参照：this.snd
- 変数：newfst,this

このように形成された FJ のプログラムに対して型検査の処理を行っていく．

4 FJ の型規則の turnstile による実装

turnstile 上に実装するにあたり，項，メソッド，クラスの３つに型付けを行う必要がある．項の型付けでは種類が多いために代表的な一つのみ以下に示す．

```
Γ |- t0 : C0  fields(C0) = C F
------------------------------
Γ |- t0.fi :Ci
```

最下行の判断は「環境Γにおいて fi というフィールドの項 t0 は Ci という型を持つ」と読む．その際には上の判断が必要条件となり，フィールド一つ一つに型を付け，そのフィールド内における項にも型を付ける必要がある．

メソッド宣言の型付け規則では M OK in C という形式を持ち「メソッド宣言 M はそれがクラス C の中にあるなら正しい形式である」と判断するものである．

```
x:C, this:C|- t0 : E0
E0 <:C0
CT(C) = class C extends D {...}
override(m,D,C → C0)
------------------------------
C0 m(C x) {return t0;} OK in C
```

このコード中の CT はクラス表を表しており，クラス名 C からクラス宣言への写像である．また述語 override では引数の型が C で結果の型が C0 であるメソッド m を D のサブクラスに定義可能かどうかの判断を行っている．

最後にクラス宣言の規則では CL OK という判断を持ちクラス宣言 CL は正しいと読むことができる．

```
K = C(D g, C f)
{super(g);this.f=f;}
fields(D) = D g  M OK in C
----------------------------------
class C extends D{C f;K M} OK
```

このように項，メソッド，クラスの型をスーパークラスからつけていくことにより turnstile 上で FJ の実装が可能になる．実際のプログラム上では今回説明をしたフィールドに関する型付けだけでなく変数参照，オブジェクト生成，キャスト等々の型付けに関する別の規則も用いることにより，型検査が可能になる．

5 まとめ

本研究ではオブジェクト指向言語の型検査を Racket 言語の turnstile ライブラリを用いて実装した．型検査はプログラム実装の際に欠かせないものとなってくる．今回のソースコードを活用することで別言語の実装を行うことが簡易的になることが分かる．また型検査だけではなく別の項目についての実装を行うことで様々な言語の実装が可能になりプログラム言語の幅が広くなる．別項目や別言語の新たな実装が今後の課題である．

参考文献

[1] Stephen Chang, Alex Knauth, and Ben Greenman. Type systems as macros. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017), 694–705, DOI:10.1145/3093333.3009886, 2017.

[2] Benjamin C. pierce，住井英二郎．型システム入門 プログラミング言語と型の理論，オーム社，2013.