ソフトウェア構成特論 第13回

大学院理工学研究科 電気電子情報工学専攻　篠埜 功

1　はじめに

これまでの計算体系に部分型 (subtyping) を導入する。その後、Java の核となる部分の型付けに関して議論するために Featherweight Java を紹介する。

2　部分型

これまでの型システムだと $(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\mathtt{r.x})\{\mathtt{x} = 0, \mathtt{y} = 1\}$ のようなプログラムは型が付かない。しかし、ラムダ抽象 $\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\mathtt{r.x}$ は引数がフィールド x を持つレコードを要求しているだけなので、引数のレコードに y などの他のフィールドがあっても問題ないはずである。また、このラムダ抽象の型により、ラムダ抽象の本体で引数のレコードの x 以外のフィールドは使わないということが分かるようにできるはずで、その場合、{x:Nat,y:Nat} 型の term は x:Nat 型を引数に要求する関数にいつでも引数として渡すことができる。

部分型 (Subtyping) を導入する目的は、上記のようなことができるようにすることである。型 S が型 T の部分型である（あるいは型 T が型 S の supertype である）ということを S <: T と書く。

これは、S 型の任意の term が T 型の term が期待されているところで使えるという意味である。別の見方としては、S 型の term の集合は T 型の term の集合の部分集合であると見てもよい。

これまでの型付け規則に、以下の subsumption と呼ばれる規則を追加する。

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{S} \quad \mathtt{S} <: \mathtt{T}}{\Gamma \vdash \mathtt{t} : \mathtt{T}} \text{ (T-SUB)}$$

S <: T なら S 型の任意の term t は T 型の term でもあるということである。例えば、{x:Nat,y:Nat} <: {x:Nat} が成り立つように subtype 関係が定義された場合には、

$$\dfrac{\dfrac{\cdots}{\vdash \{\mathtt{x} = 0, \mathtt{y} = 1\} : \{\mathtt{x} : \mathtt{Nat}, \mathtt{y} : \mathtt{Nat}\}} \text{ (T-RCD)} \quad \dfrac{\cdots}{\{\mathtt{x} : \mathtt{Nat}, \mathtt{y} : \mathtt{Nat}\} <: \{\mathtt{x} : \mathtt{Nat}\}} \, (\cdots)}{\vdash \{\mathtt{x} = 0, \mathtt{y} = 1\} : \{\mathtt{x} : \mathtt{Nat}\}} \text{ (T-SUB)}$$

により、型判定 $\vdash \{\mathtt{x} = 0, \mathtt{y} = 1\} : \{\mathtt{x} : \mathtt{Nat}\}$ が成立する。

3 部分型関係 (subtype relation)

部分型関係 (subtype relation) は、S <: T の形の statement を導出するための推論規則により定義される。S <: T は、「S は T の部分型である (S is a subtype of T)」と読む。

まず、部分型関係は反射的であるべきなので、

$$\frac{}{\mathtt{S} <: \mathtt{S}}\ \text{(S-REFL)}$$

を追加する。また、部分型関係は推移的であるべきなので、

$$\frac{\mathtt{S} <: \mathtt{U} \quad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}}\ \text{(S-TRANS)}$$

を追加する。

また、レコード式の間の *width subtyping* 規則を追加する。

$$\frac{}{\{\mathtt{l}_i\colon \mathtt{T}_i^{i\in 1\ldots n+k}\} <: \{\mathtt{l}_i\colon \mathtt{T}_i^{i\in 1\ldots n}\}}\ \text{(S-RCDWIDTH)}$$

つまり、フィールドが多い方が型が小さい。例えばレコード型 {x:Nat} は、Nat 型のフィールド x を持つレコードすべての集合と考えればよい。{x=3} や {x=5} はこの型の要素であり、{x=3,y=100} や {x=3,a=true,b=true} もこの型の要素である。同様に、レコード型 {x:Nat,y:Nat} は、Nat 型のフィールド x, y を持つレコードすべての集合と考えればよい。{x=3,y=100} や {x=3,y=100,z=true} はこの型の要素であるが、{x=3} や {x=3,a=true,b=true} はこの型の要素ではない。つまり、{x:Nat,y:Nat} 型の term 集合は {x:Nat} 型の term 集合の真の部分集合 (proper subset) である。

Width subtyping 規則 (S-RCDWIDTH) は、ラベルが同じフィールドが同じ型の場合のみ使える。ラベルが同じフィールドで、それらの型が部分型の関係であればレコード間に部分型関係があるようにしたいので、次の *depth subtyping* 規則を追加する。

$$\frac{\text{for each } i \quad \mathtt{S}_i <: \mathtt{T}_i}{\{\mathtt{l}_i\colon \mathtt{S}_i^{i\in 1\ldots n}\} <: \{\mathtt{l}_i\colon \mathtt{T}_i^{i\in 1\ldots n}\}}\ \text{(S-RCDDEPTH)}$$

例えば、レコード型 {x:{a:Nat,b:Nat},y:{m:Nat}} はレコード型 {x:{a:Nat},y:{}} の部分型であり、その導出木は以下の通りである。

$$\frac{\dfrac{}{\{\mathtt{a}:\mathtt{Nat},\mathtt{b}:\mathtt{Nat}\} <: \{\mathtt{a}:\mathtt{Nat}\}}\ \text{(S-RCDWIDTH)} \quad \dfrac{}{\{\mathtt{m}:\mathtt{Nat}\} <: \{\}}\ \text{(S-RCDWIDTH)}}{\{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat},\mathtt{b}:\mathtt{Nat}\},\mathtt{y}:\{\mathtt{m}:\mathtt{Nat}\}\} <: \{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat}\},\mathtt{y}:\{\}\}}\ \text{(S-RCDDEPTH)}$$

レコードの 1 つのフィールドのみが（真の）部分型になる場合は、S-REFL 規則を使えばよい。

$$\frac{\dfrac{}{\{\mathtt{a}:\mathtt{Nat},\mathtt{b}:\mathtt{Nat}\} <: \{\mathtt{a}:\mathtt{Nat}\}}\ \text{(S-RCDWIDTH)} \quad \dfrac{}{\{\mathtt{m}:\mathtt{Nat}\} <: \{\mathtt{m}:\mathtt{Nat}\}}\ \text{(S-REFL)}}{\{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat},\mathtt{b}:\mathtt{Nat}\},\mathtt{y}:\{\mathtt{m}:\mathtt{Nat}\}\} <: \{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat}\},\mathtt{y}:\{\mathtt{m}:\mathtt{Nat}\}\}}\ \text{(S-RCDDEPTH)}$$

また、width, depth subtyping を結合させる場合は例えば以下のように S-TRANS 規則を使えばよい。

$$\frac{\text{(*1)} \qquad \text{(*2)}}{\{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat},\mathtt{b}:\mathtt{Nat}\},\mathtt{y}:\{\mathtt{m}:\mathtt{Nat}\}\} <: \{\mathtt{x}:\{\mathtt{a}:\mathtt{Nat}\}\}}\ \text{(S-TRANS)}$$

ここで (*1) の部分は

$$\frac{}{\{\mathtt{x} : \{\mathtt{a} : \mathtt{Nat}, \mathtt{b} : \mathtt{Nat}\}, \mathtt{y} : \{\mathtt{m} : \mathtt{Nat}\}\} <: \{\mathtt{x} : \{\mathtt{a} : \mathtt{Nat}, \mathtt{b} : \mathtt{Nat}\}\}} \text{ (S-RCDWIDTH)}$$

であり、(*2) の部分は

$$\frac{\dfrac{}{\{\mathtt{a} : \mathtt{Nat}, \mathtt{b} : \mathtt{Nat}\} <: \{\mathtt{a} : \mathtt{Nat}\}} \text{ (S-RCDWIDTH)}}{\{\mathtt{x} : \{\mathtt{a} : \mathtt{Nat}, \mathtt{b} : \mathtt{Nat}\}\} <: \{\mathtt{x} : \{\mathtt{a} : \mathtt{Nat}\}\}} \text{ (S-RCDDEPTH)}$$

である。

レコードに対する演算は projection のみであり、それには他にどのようなフィールドがあるかは関係ないので、フィールドの順番が入れ替わったレコード型はもとのレコード型の部分型になるようにしたい。そこで以下の規則を導入する。

$$\frac{\{\mathtt{k}_j : \mathtt{S}_j^{j \in 1 \ldots n}\} \text{ is a permutation of } \{\mathtt{l}_i : \mathtt{T}_i^{i \in 1 \ldots n}\}}{\{\mathtt{k}_j : \mathtt{S}_j^{j \in 1 \ldots n}\} <: \{\mathtt{l}_i : \mathtt{T}_i^{i \in 1 \ldots n}\}} \text{ (S-RCDPERM)}$$

例えば、レコード型 {c:Nat,b:Bool,a:Nat} はレコード型 {a:Nat,b:Bool,c:Nat} の部分型であり、レコード型 {a:Nat,b:Bool,c:Nat} はレコード型 {c:Nat,b:Bool,a:Nat} の部分型である。つまり、ここまで挙げた規則により定義される部分型関係<:は反対称的 (anti-symmetric) ではない。

補足: $\forall x, y \in A.\ xRy \wedge yRx \Rightarrow x = y$ が成り立つ二項関係 R を反対称的な関係という。

S-RCDPERM 規則, S-RCDWIDTH 規則, S-TRANS 規則により、任意のレコード型 T について、任意のフィールドを除去したレコード型 T' について、T <: T' が成立する。(除去したフィールドを T 型のレコード（除去してない方）で (S-RCDPERM) 規則で最後に移動し、順番を入れ替えた T 型と T' 型を (S-RCDWIDTH) 規則で比較すれば良い。その際、それら 2 つの規則の適用を (S-TRANS) 規則で結合させれば良い。)

ここまでの規則はレコード型の部分型関係を導く規則である。次に関数型の間の部分型関係を導く規則を導入する。

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \quad \mathtt{S}_2 <: \mathtt{T}_2}{\mathtt{S}_1 \to \mathtt{S}_2 <: \mathtt{T}_1 \to \mathtt{T}_2} \text{ (S-ARROW)}$$

引数の型について、部分型関係が逆 (contravariant) になり、結果の型については同じ向き (covariant) である。これを、以下のように表現する。

> 関数型の型構成子 → は、引数の型について contravariant であり、結果の型について covariant である。

直感的には、$\mathtt{S}_1 \to \mathtt{S}_2$ 型の関数 f は、$\mathtt{S}_1$ 型の引数や、$\mathtt{S}_1$ 型の部分型である $\mathtt{T}_1$ 型の引数も受け取ることができる。また、関数 f の返り値の型は $\mathtt{S}_2$ であるが、これは、$\mathtt{S}_2$ 型の supertype である $\mathtt{T}_2$ 型にも属している。つまり、$\mathtt{S}_1 \to \mathtt{S}_2$ 型の関数は、$\mathtt{T}_1 \to \mathtt{T}_2$ 型も持つ。(別の言い方をすれば、$\mathtt{S}_1 \to \mathtt{S}_2$ 型の関数は、$\mathtt{T}_1 \to \mathtt{T}_2$ 型が期待されているところで使える。)

補足: Java では、配列の型構成子 [] は、配列の要素の型について covariant である。つまり、A <: B のとき、A [] <: B [] である。これは問題があり、Java の設計の失敗と考えられている。

```
class A {...}
class B extends A {...}
B [] bArray = new B[10];
A [] aArray = bArray;
aArray[0] = new A();
```

のような場合に、`aArray[0] = new A();` の部分で実行時に `ArrayStoreException` 例外が発生する。今は Generics が Java に入っており、この問題は回避できる。

最後に、あらゆる型の supertype になる型があると便利がよい。これを `Top` 型とし、次の規則を追加する。

$$\frac{}{\mathtt{S} <: \mathtt{Top}}\ \text{(S-TOP)}$$

形式的には、部分型関係は、これまでに提示した規則のもとで閉じている最小の関係である。S-REFL 規則と S-TRANS 規則により、部分型関係は preorder であるが、S-RCDPERM 規則があるので、partial order ではない (反対称的ではないので)。

補足 1: 反射的 (reflexive) かつ推移的 (transitive) な二項関係を preorder という。

補足 2: 反射的 (reflexive) かつ推移的 (transitive) かつ反対称的 (anti-simmetric) な二項関係を partial order という。

例 1. 最初に挙げた例 $(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\mathtt{r.x})\{\mathtt{x} = 0, \mathtt{y} = 1\}$ がこれまでの規則により型を持つことを示す。

$$\frac{(*1) \quad (*2)}{\vdash (\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\mathtt{r.x})\ \{\mathtt{x} = 0, \mathtt{y} = 1\}\ :\ \mathtt{Nat}}\ \text{(T-APP)}$$

(*1) の部分は以下の導出木である。

$$\frac{\dfrac{\dfrac{\mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\} \in \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}}{\mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\} \vdash \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}}\ \text{(T-VAR)}}{\mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\} \vdash \mathtt{r.x} : \mathtt{Nat}}\ \text{(T-PROJ)}}{\vdash \lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\mathtt{r.x} : \{\mathtt{x} : \mathtt{Nat}\} \to \mathtt{Nat}}\ \text{(T-ABS)}$$

(*2) の部分は以下の導出木である。

$$\frac{\dfrac{\dfrac{}{\vdash 0 : \mathtt{Nat}}\ \text{(T-ZERO)} \quad \dfrac{\dfrac{}{\vdash 0 : \mathtt{Nat}}\ \text{(T-ZERO)}}{\vdash 1 : \mathtt{Nat}}\ \text{(T-SUCC)}}{\vdash \{\mathtt{x} = 0, \mathtt{y} = 1\} : \{\mathtt{x} : \mathtt{Nat}, \mathtt{y} : \mathtt{Nat}\}}\ \text{(T-RCD)} \quad \dfrac{}{\{\mathtt{x} : \mathtt{Nat}, \mathtt{y} : \mathtt{Nat}\} <: \{\mathtt{x} : \mathtt{Nat}\}}\ \text{(S-RCDWIDTH)}}{\vdash \{\mathtt{x} = 0, \mathtt{y} = 1\} : \{\mathtt{x} : \mathtt{Nat}\}}\ \text{(T-SUB)}$$

練習問題 1. 型判定 $\vdash (\lambda \mathtt{r} : \{\mathtt{y} : \mathtt{Nat}\}.\mathtt{r.y})\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Nat}$ の導出木を示せ。

練習問題 2. term $(\lambda \mathtt{r} : \{\mathtt{y} : \mathtt{Nat}\}.\mathtt{r.y})\{\mathtt{x} = 0, \mathtt{y} = 1\}$ を評価せよ。各 1 ステップ評価の導出木も書け。

4 部分型関係の性質

補題 1 (Inversion of the subtype relation)**.**

1. $\mathtt{S} <: \mathtt{T}_1 \to \mathtt{T}_2$ ならば $\mathtt{S}$ は $\mathtt{S}_1 \to \mathtt{S}_2$ の形をしており、$\mathtt{T}_1 <: \mathtt{S}_1$ かつ $\mathtt{S}_2 <: \mathtt{T}_2$ である。
2. $\mathtt{S} <: \{\mathtt{l}_i : \mathtt{T}_i^{i\in 1\ldots n}\}$ ならば $\mathtt{S}$ は $\{\mathtt{k}_j : \mathtt{S}_j^{j\in 1\ldots m}\}$ という形をしており、$\{\mathtt{l}_i^{i\in 1\ldots n}\} \subseteq \{\mathtt{k}_j^{j\in 1\ldots m}\}$ であり、$\mathtt{l}_i = \mathtt{k}_j$ のとき $\mathtt{S}_j <: \mathtt{T}_i$ である。

証明. 省略する。 □

補題 2.

1. $\Gamma \vdash \lambda \mathtt{x} : \mathtt{S}_1.\mathtt{s}_2 : \mathtt{T}_1 \to \mathtt{T}_2$ ならば $\mathtt{T}_1 <: \mathtt{S}_1$ かつ $\Gamma, \mathtt{x} : \mathtt{S}_1 \vdash \mathtt{s}_2 : \mathtt{T}_2$ である。
2. $\Gamma \vdash \{\mathtt{k}_a = \mathtt{s}_a^{a\in 1\ldots m}\} : \{\mathtt{l}_i : \mathtt{T}_i^{i\in 1\ldots n}\}$ ならば $\{\mathtt{l}_i^{i\in 1\ldots n}\} \subseteq \{\mathtt{k}_a^{a\in 1\ldots m}\}$ かつ $\mathtt{k}_a = \mathtt{l}_i$ のとき $\Gamma \vdash \mathtt{s}_a : \mathtt{T}_i$ である。

証明. 省略する。 □

補題 3 (Substitution)**.** $\Gamma, \mathtt{x}, \mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ かつ $\Gamma \vdash \mathtt{s} : \mathtt{S}$ ならば $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$ である。

証明. 省略する。 □

定理 1 (Preservation)**.** $\Gamma \vdash \mathtt{t} : \mathtt{T}$ かつ $\mathtt{t} \to \mathtt{t}'$ ならば $\Gamma \vdash \mathtt{t}' : \mathtt{T}$ である。

証明. 省略する。 □

定理 2 (Progress)**.** $\mathtt{t}$ が閉じた term であり、型がつく term であるとき、$\mathtt{t}$ は値であるか、あるいはある $\mathtt{t}'$ が存在して $\mathtt{t} \to \mathtt{t}'$ である。

証明. 省略する。 □

5 Featherweight Java

Java の核となる部分 (Featherweight Java, 略して FJ) に対し、操作的意味論、型システムを（次回）与える。その導入としてここでは FJ の概略を説明する。FJ の目的は、Java をできる限り削って Java プログラムの型付けの核となる部分を示すことである。

FJ のプログラムは、クラス定義の集まりと、評価対象の 1 つの term (Java では main メソッドの本体に相当) からなる。term はオブジェクトの生成、メソッド呼び出し、フィールドアクセス、キャスト、変数の 5 つしかない。

FJ のプログラムは以下のようなものである。

```
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); } }
```

FJ では、superclass を (Object の場合でも) 必ず書く。また、constructor も必ず書く。また、フィールドアクセスやメソッド呼び出しは、term. フィールド名、term. メソッド名 (...) の形で (term が this の場合でも) 書く。constructor は、各フィールドに対し、フィールド名と同じ名前の仮引数をフィールドの宣言と同じ順番で取り、最初に super により親クラスのフィールドの初期化を行う。上記の例では、クラス A, B, C の親クラスは Object である。Object クラスはフィールドを持たず、super の呼び出しは引数がない。super や =は constructor 内にのみ現れうる。FJ は副作用がなく、メソッドの本体は常に return 文からなる。

上記のクラス定義において、term

```
new Pair(new A(), new B()).setfst(new B())
```

は new Pair(new B(), new B()) に評価される。また、別の例では、term

```
((Pair) (new Pair(new Pair(new A(), new B()),
                  new A()).fst)).snd
```

は new B() に評価される。この式の中の

```
(Pair) (new Pair(new Pair(new A(), new B()), new A()).fst)
```

の部分は cast 式である。この中の

```
new Pair(new Pair(new A(), new B()), new A()).fst
```

は fst というフィールドへのアクセスの式であり、fst というフィールドは Object 型として宣言されている。それにもかかわらず、次は snd というフィールドにアクセスしているので、Pair 型でなければならない。よって (Pair) でキャストをしている。実行時に、評価規則適用時に fst フィールドの Object 型のオブジェクトが Pair 型のオブジェクトであることを検査する（この例では検査は成功する)。

次の例は、フィールドアクセスに対する評価規則 (E-PROJNEW)（第 14 回資料参照）の適用例を示す。

$$\texttt{new Pair(newA(), new B()).snd} \rightarrow \texttt{new B()}$$

Pair クラスでは snd は 2 番目に宣言されているので、constructor の 2 番目の引数である new B() が結果となる。

次の例は、メソッド呼び出しに対する評価規則 (E-INVKNEW)（第 14 回資料参照）の適用例を示す。

$$\begin{aligned}&\texttt{new Pair(new A(), new B()).setfst(new B())}\\ &\rightarrow [\texttt{newfst} \mapsto \texttt{new B()}, \texttt{this} \mapsto \texttt{new Pair(new A(), new B())}]\texttt{new Pair(newfst, this.snd)}\end{aligned}$$

結果の部分の置換を行った term は、

```
new Pair(new B(), new Pair(new A(), new B()).snd)
```

である。setfst メソッド呼び出しを受け取るオブジェクトは、new Pair(new A(), new B()) なので、Pair クラスの setfst メソッドの定義を見る。その仮引数が newfst で本体が new Pair(newfst, this.snd) である。メソッド呼び出しは、その本体の仮引数部分が実引数で置き換えられたものに評価される。その際、this はメソッドを受け取るオブジェクトに置き換えられる。これはラムダ計算の β 簡約に似ている。FJ では、ラムダ計算同様、メソッドの本体内に同じ仮引数が 2 回以上現れる場合、実引数が複製されることになるが、FJ では副作用がないので Java の意味との違いは外からは見えない。

次の例は、cast に対する評価規則 (E-CASTNEW)（第 14 回資料参照）の適用例を示す。

$$(\texttt{Pair})\texttt{new Pair}(\texttt{new A}(),\ \texttt{new B}()) \rightarrow \texttt{new Pair}(\texttt{new A}(),\ \texttt{new B}())$$

cast の対象の term がオブジェクトまで評価されたら、constructor が作るオブジェクトのクラスが cast 先のクラスのサブクラスであることを確認する。もし確認が成功すれば、cast が外されたものが評価結果となる。もし失敗すれば、計算が stuck する。この例の場合、同じクラスなので、サブクラスでもあり、この確認は成功し、単にキャストが外されたものが結果となる。

評価が stuck する場合は 3 通りある。宣言されていないフィールドへのアクセス、宣言されていないメソッドの呼び出し、オブジェクトの（実行時の）クラスの superclass 以外へのキャストの 3 つである。型がついた FJ のプログラムにおいては、このうちの最初の 2 つは起こらないことを証明できる。3 つ目の superclass 以外へのキャストは、downcast と stupid cast である。（第 14 回資料参照）superclass へのキャストは up cast と言い、問題がない。

FJ の評価戦略は通常の値呼び (call by value) である。さきほどの cast の例を値まで評価すると以下のようになる。

$$\begin{aligned}
& ((\texttt{Pair})\ (\texttt{new Pair}(\texttt{new Pair}(\texttt{new A}(), \texttt{new B}()),\ \texttt{new A}()).\texttt{fst})).\texttt{snd} \\
\rightarrow\ & ((\texttt{Pair})\ (\texttt{new Pair}(\texttt{new A}(), \texttt{new B}()))).\texttt{snd} \\
\rightarrow\ & (\texttt{new Pair}(\texttt{new A}(), \texttt{new B}())).\texttt{snd} \\
\rightarrow\ & \texttt{new B}()
\end{aligned}$$