Generics・Union型を導入したオブジェクト指向計算体系

柳楽 秀士　　五十嵐 淳
京都大学大学院情報学研究科

概 要

Java は型安全な言語と言われているが，実際には，汎用的なデータ構造やデータの和集合を表現する際に，動的検査を伴なう型変換機構に頼ってプログラムを書く場面が多い．このような型変換の用法は，型システムが柔軟性を欠くために強いられたものであり，安全性・効率の面から好ましくないものである．本稿では，この問題を解決するためのより安全な機構—具体的には，generics による型抽象と，和集合を表す union 型を併せ持つ型システム— を提案する．また，このような言語機構の安全性を示すため，Igarashi, Pierce, Wadler による計算体系 FGJ に union 型を導入した体系 FGJ∨ の構文，型付け規則，操作的意味の形式的定義を与え，型健全性を証明する．

1　はじめに

クラスに基づくオブジェクト指向言語においてはクラスの継承によってコードの再利用やオブジェクトの多態性などの機能を実現している．しかしクラスの継承のみではある種のプログラムイディオムの表現には不十分であり，表現の拡張が必要である．

拡張の一つとしてリスト・木などの汎用的データ構造を表現するための，汎用クラス (generics) と呼ばれる，クラス定義を型情報でパラメータ抽象化する機構が提案されており [14, 4, 8, 3, 16, 7, 17, 1] Java 言語にも次期バージョンで GJ [3] に基づく generics 機構が導入される予定である．さらにデータの和集合を表す型として著者らは union 型 [22] を提案している．

これらの機構が備わっていない言語でも，型変換 (キャスト) 機構を用いて同様なプログラミングが可能ではあるが，我々は，これらふたつの機構は，オブジェクト指向プログラムの安全性・再利用性を高めるために非常に重要だと考える．そこで，本研究では，これらふたつの型機構を同時に併せ持つ言語が型安全であることを示す．そのために，generics (より正確には Java の拡張である GJ [3]) のための形式モデルである Featherweight GJ (FGJ) [12] に， union 型を加え拡張した核言語 FGJ∨ を設計し，その構文，型付け規則，操作的意味の形式的定義を与え，型健全性を証明する．

本論文の構成は以下の通りである．第 2 節では，generics と union 型の概要をプログラム例とともに示す．第 3 節で FGJ∨ の形式的定義を与え，第 4 節で FGJ∨ の満たす性質について述べる．最後に関連研究との比較を第 5 節で行い，第 6 節で結論を述べる．本文中では主に，体系の形式的定義の一部と成立する定理とその証明概略について述べるにとどめ，体系全体の定義は付録とし，定理の詳しい証明については省略する．詳しくは，第一著者の修士論文 [21] を参照されたい．

2 Generics と Union 型

本節では，汎用クラスや union 型を使ったプログラミングの概要と，それにより，どのようなダウンキャストの典型的な使用例が不要になるか (より自然な記述になるか) を概観する．以下のプログラム例は，本論文で導入される FGJ∨ ひいては GJ を元にした文法で書かれている．

2.1 汎用データ構造のための Generics

汎用クラス (generics) は，C++ のテンプレートに見られるように，クラス定義の型情報の一部を変数としてパラメータ抽象化したものである．そのパラメータを違う型で具体化することで，様々なクラス定義を導くことができる．汎用クラスは，リスト・木などの汎用データ構造の表現に適している．

例えば，リスト構造のための汎用クラスは以下のように定義できる．

```
class List<X> {
  X car;  List<X> cdr;
  ...
}
```

ここで X は型上を動く変数で型変数と呼ばれる．汎用クラス名 List の直後で宣言することで，クラス定義内で型として使用できる．この例ではフィールド car，つまり，リストの要素の型を X として宣言している．List 型を参照する時には，cdr の型のように，明示的に要素の型を与え，List<要素型> のような形で記述する．(ここでは，型変数 X が要素型として使われている．)

型検査は List に与えられた具体的な型に従って行われ，以下の例のように，List<Integer> の要素として Integer，List<String> の要素として String の値を取ることができる．

```
// Integer のリストとして扱う
List<Integer> l1 = new List<Integer>(new Integer(10),
                                     new List<Integer>(new Integer(20),null));

Integer i = l1.car;

// String のリストとして扱う
List<String> l2 = new List<String>("aa",
                                   new List<String>("bb",null));
String s = l2.car;
```

もちろん違う型の値を入れると型検査は失敗する．

```
// Error!! (Integer のリストの要素として String は取れない)
List<Integer> list = new List<Integer>(new String("aa"),null);
```

このため List<Integer> 型のフィールド car は Integer 型であることが保証される．

一方，これを現在の Java など，汎用クラスを持たない言語で記述すると，

```
class List {
  Object car;  List cdr;
  ...
}
```

となる．型変数に相当する部分は，全ての (参照) 型が部分型である Object が使われているため，要素の挿入は以下のように，部分型による暗黙の型変換を用いて行える．

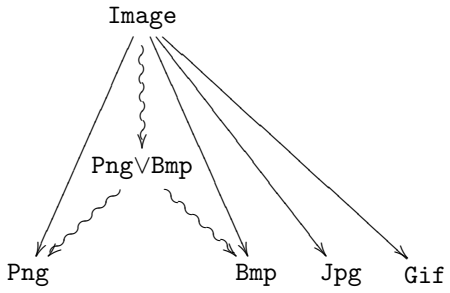

図 1: Union 型による部分型関係

```
List l1 = new List(new Integer(10),
                   new List(new Integer(20),null));
```

しかし，要素を取り出す際には，car の型は Object であるために，プログラマは要素の型を推察しダウンキャストを行うことになる．

```
Integer i = (Integer)l1.car;
```

こういったプログラミングは，人間であるプログラマのプログラムに関する知識・推論に依存しており，型検査が保証するプログラムの安全性は汎用クラスを用いる場合に比べ，低下していると考えられる．例えば，次のようなプログラムは型検査を通過するが実行時にエラーとなる．

```
List l1 = new List(new String("aa"),
                   new List(new Integer(20),null));

Integer i = (Integer)l1.car; // String 型の値を Integer に変換している
```

もちろんプログラマはこのような実行時エラーが起きないように注意してプログラムを記述するが，上のようなエラーの可能性があるため実行時検査が必要となる．プログラム解析を行うことで省略できる実行時検査もあるとはいえ，実行時に検査が行われてしまうとしたら効率面でも好ましくない．[1]

2.2 異なる型のデータをまとめて扱うための Union 型

我々は，最近の研究 [22] で union 型という機構を提案している．具体的には，型表現として C∨D という記法を導入することで，ふたつの型 C と D に属するインスタンス集合の和を表現することができる．union 型それ自体は，インスタンスを持たない (new できない) 型で，部分型関係においては，ふたつの型の上限 (ふたつの型を部分型とする型のうち最小のもの) に位置するものである．例えば，図 1 は画像データを表現する型の部分型関係を表している．実線は継承による部分型関係，波線が union 型による部分型関係であり，Png∨Bmp は Png と Bmp の上，Image より下に位置する．ここで注目すべきは，同じ Image から派生した他の型 Jpg や Gif とは部分型関係を持たないことである．

この型を使って次のように Png と Bmp の値をまとめて扱うことができる．

[1] 現在の GJ の実装では，ここに示した汎用クラスのプログラムは，ダウンキャストを使うプログラムに変換されてしまうため，効率面での向上はない．

```
Png∨Bmp x = new Png("foo.png");
...
case x of
  (Png y) { Png の場合の動作; }
| (Bmp z) { Bmp の場合の動作; }
```

ここで現れる case 文は x の値の型を実行時に検査し，Png として扱える型 (Png の部分型) であるとき Png として扱い， Bmp の場合には Bmp として扱い処理を行う文である．(静的) 型検査では，x の型が Png または Bmp として扱える型である (つまり，x の型が Png∨Bmp の部分型である) ことを検査すれば，どちらかの場合の処理が，必ず実行されることを保証できると考えられる．またこのように表現することで動作の定義が分散してしまうことを防ぐこともできる．

これを union 型 を使わないで，共通の super type である Image を使って，

```
Image x = new Png("foo.png");
...
if(x instanceof Png){
  (Png)x を使って Png の場合の動作を記述
} else {
  (Bmp)x を使って Bmp の場合の動作を記述
}
```

と表現することもできる．しかし Image x = new Jpg("bar.jpg"); のように，Image 型の変数 x に Png または Bmp 以外の値が束縛されることは静的に検査することができない．この場合 Jpg の値を Bmp として扱おうとして実行時にエラーが発生する．

我々の提案では，場合分けによる操作以外にも，union 型の変数・式に対して直接—場合分けすることなく—(virtual) メソッド呼出しやフィールドの読み書きを許している．つまり，C∨D という型に対し，C と D が共に持つメンバに直接アクセスが可能である．共に持つメンバとは，C と D の共通の親クラスから継承された定義だけではなく，独立して導入された同名のメンバをも含むものである．この機能を応用すると，union 型でグループ化されるクラスに共通のメソッドを追加し，それを virtual メソッド呼び出しできる．例えば Png と Bmp に新たなメソッドを加えることを考える．Java でこのようなプログラミングを行おうとする場合，典型的には newMethod を持つインターフェース MyPngOrMyBmp を宣言し，MyPng, MyBmp はそれを implement し，img の型として MyPngOrMyBmp を使うことになる．

```
interface PngOrBmp{
  void newMethod(); // 共通のメソッドを定義することを強要する
  void draw();      // Image に定義されたメソッド
}
class MyPng extends Png implements PngOrBmp {
  void newMethod(){ Png の場合の処理; }
}
class MyBmp extends Bmp implements PngOrBmp {
  void newMethod(){ Bmp の場合の処理; }
}

MyPngOrMyBmp img = new MyPng("foo.png");

img.newMethod();
imp.draw();      // Image から継承されたメソッド呼出し.
```

ただし，このインターフェースは Image の部分型ではないため Image から継承したメソッド draw を 呼ぶためには MyPngOrMyBmp で明示的に draw を宣言しなくてはならない．しかし，union 型を利用すると，単に新しいメソッドを宣言し，ふたつの型の union を構成することで実現できる．

```
class MyPng extends Png { void newMethod(){ Png の場合の処理; } }
class MyBmp extends Bmp { void newMethod(){ Bmp の場合の処理; } }

MyPng∨MyBmp img = new MyPng("foo.png");

img.newMethod(); // 同名のメソッド呼出し
img.draw(); // MyPng,MyBmp が Image から継承したメソッド
```

ここで注目すべきは，予め newMethod というメソッドを持つような共通のインターフェースを用意しなくても，継承によって機能を追加し，union 型 MyPng∨MyBmp を記述するだけで，MyPng, MyBmp のインターフェースの共通部分を抽出したような効果が得られていることである．

このように，union 型を使用する利点としては，

- 複数の型の和集合を構成し，場合分けでいずれかの分岐が実行されることが静的に検査できる
- いわば「後付け」で，複数の型に共通するインターフェイスを抽出できる

ことなどが挙げられる．

2.3 Generics，Union の併用

本節の最後の例として，汎用クラスと union 型を併用する例について述べる．以下のように，異なる種類のデータが混在するデータ構造を汎用クラス List の型引数として union 型を与えることにより，自然に記述することができる．

```
class List<X> {
  X car;  List<X> cdr;
  ...
}

List<Png∨Bmp> l = new List<Png∨Bmp>(new Png("foo.png"),
                                    new List<Png∨Bmp>(new Bmp("bar.bmp"),null));

case l.car of
    (Png png) { Png の場合の処理; }
  | (Bmp bmp) { Bmp の場合の処理; }
```

以上のように，汎用データ構造の記述のための generics，和集合記述のための union 型，そして，その併用により，より抽象度が高く，安全なプログラミングが可能であるが，これらが共存する言語の意味論や型システムの安全性は議論されてきていない．次節では，これらの機構を導入した形式的体系 FGJ∨ を定義し，その型健全性，特に case による場合分けが完全であることや，共通メンバの呼び出しが安全に行えることを示すことによって体系の安全性を確認する．

3 FGJ∨

本節で導入する体系 FGJ∨ は，GJ の核部分を形式化した計算体系 Featherweight GJ (FGJ)[12] を拡張したものである．FGJ は Java などのクラスに基づくオブジェクト指向言語をモデル化した体系であり，クラスや継承，(virtual) メソッド呼び出し，フィールド，this といった最小限の機能と generics をモデル化している． FGJ∨ はこれを union 型・case 構文によって拡張した体系となっている． (ただし，FGJ の型変換機構は case の導入に伴ない除いている．)

以下では，主要な部分の定義を示し説明を行う．体系全体の定義は付録 A に示す．

3.1 文法

以下で，C，D はクラス名，X，Y，Z は型変数，x, y は変数，f，g はフィールド名，m はメソッド名，e，d は式を表すメタ変数とする．型はその用途により三種類あり，new によりインスタンスを生成したり親クラスとして使える C<$\overline{\mathtt{T}}$> の形の型を**インスタンス型**，型変数の動く範囲の上限を与えるために使えるものを**非変数型** (メタ変数 N, P)，その他，フィールドやメソッドの引数・返り値の宣言に使用できる型を単に**型**(メタ変数 S，T，U) と呼ぶ．インスタンス型は非変数型であり，非変数型は型である．

FGJ$\vee$ におけるクラスなどの文法は以下の通りである．

N	::=	C<$\overline{\mathtt{T}}$> | N$\vee$N	非変数型
T	::=	X | C<$\overline{\mathtt{T}}$> | T$\vee$T	型
L	::=	class C<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$>$\triangleleft$C<$\overline{\mathtt{T}}$> {$\overline{\mathtt{T}}$ $\overline{\mathtt{f}}$; $\overline{\mathtt{M}}$}	クラス定義
M	::=	<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> T m($\overline{\mathtt{T}}$ $\overline{\mathtt{x}}$){ return e; }	メソッド定義
e	::=	x | this | e.f | e.m<$\overline{\mathtt{T}}$>($\overline{\mathtt{e}}$) | new C<T>($\overline{\mathtt{e}}$)	
	|	(case e of (T x) e | (T x) e)	式
v	::=	new C<$\overline{\mathtt{T}}$>($\overline{\mathtt{v}}$)	値

$\overline{\mathtt{f}}$ のような上線付きのメタ変数は列を表す．例えば，$\overline{\mathtt{f}}$ は $\mathtt{f}_1,\mathtt{f}_2,\ldots,\mathtt{f}_n$ を，$\overline{\mathtt{T}}$ $\overline{\mathtt{f}}$ は $\mathtt{T}_1$ $\mathtt{f}_1,\ldots,\mathtt{T}_n$ $\mathtt{f}_n$ を，$\overline{\mathtt{T}}$ $\overline{\mathtt{f}}$; は $\mathtt{T}_1$ $\mathtt{f}_1;\cdots\mathtt{T}_n$ $\mathtt{f}_n$; を表す．$n \geq 0$ であり $n = 0$ の場合は空列を表すものとする．$\triangleleft$ は Java における extends を表す記号である．クラス定義やメソッド定義における <$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> は <$\mathtt{X}_1\triangleleft\mathtt{N}_1$, ...,$\mathtt{X}_n\triangleleft\mathtt{N}_n$> の略記であり，クラス定義やメソッド定義を抽象化する型変数 $\mathtt{X}_i$ の動く範囲が $\mathtt{N}_i$ の部分型である[2]ことを示している．ここで，GJ の設計に基づき上限としては非変数型しか使えないことに注意されたい．この制限は，GJ では X$\triangleleft$C<X> や，X$\triangleleft$C<Y>，Y$\triangleleft$D<X> といった，F-bounded polymorphism [6] と呼ばれる (相互) 再帰的な上限指定が許されていることに起因する．ここで，上限を非変数型にすることで，X$\triangleleft$X や X$\triangleleft$Y,Y$\triangleleft$X といった無意味な指定を文法的に防ぐことができる．(次に述べる束縛変数の定義でわかるように，X$\triangleleft$N において N 中の X の出現は束縛される．) 以下，$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$ が空の場合，<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> を省略することができる．インスタンス型に対しても同様に空の <> を省略し，C<> を C と書く．

(型) 変数の有効範囲は以下のように定義する．クラス定義 class C<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> $\triangleleft$ D<$\overline{\mathtt{S}}$> {$\overline{\mathtt{T}}$ $\overline{\mathtt{f}}$; $\overline{\mathtt{M}}$}, メソッド定義 <$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> $\mathtt{T}_0$ m($\overline{\mathtt{T}}$ $\overline{\mathtt{x}}$){ return e; } において $\overline{\mathtt{X}}$ は束縛されており，有効範囲はそれぞれ $\overline{\mathtt{N}}$, $\overline{\mathtt{S}}$, $\overline{\mathtt{T}}$, $\overline{\mathtt{M}}$ と $\overline{\mathtt{N}}$, $\mathtt{T}_0$, $\overline{\mathtt{T}}$, e である．型変数はその上限中での出現も束縛することに注意されたい．また，メソッド定義 <$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> $\mathtt{T}_0$ m($\overline{\mathtt{T}}$ $\overline{\mathtt{x}}$){ return e; } では $\overline{\mathtt{x}}$ は有効範囲 e で束縛され case $\mathtt{e}_0$ of ($\mathtt{T}_1$ $\mathtt{x}_1$) $\mathtt{e}_1$ | ($\mathtt{T}_2$ $\mathtt{x}_2$) $\mathtt{e}_2$ において，$\mathtt{x}_1$ と $\mathtt{x}_2$ は，それぞれ有効範囲 $\mathtt{e}_1$ と $\mathtt{e}_2$ で束縛されている．この体系において this は (キーワードではなく) 変数のひとつとして扱われ，メソッド内で暗黙のうちに束縛されているとみなされる．(メソッド仮引数や case の束縛変数としては使われないとする．) this の型は，後で定義する型付け規則からわかるように，それが現われるクラスに応じた適当な型が与えられることになる．

FGJ にみられるコンストラクタの定義は Java の文法に合わせるための形式的なものなので省略しており，各クラスは，各フィールドの初期値を引数とし，それを各フィールドに代入する自明なコンストラクタをひとつ持つと仮定する．

FGJ$\vee$ プログラムはクラスの集合 $\overline{\mathtt{L}}$ と (main メソッドに相当する) 式 e の組で与えられる．以下では，ある与えられたクラスの集合 $\overline{\mathtt{L}}$ を仮定する．さらに，$\overline{\mathtt{L}}$ の継承関係には循環がなく，$\overline{\mathtt{L}}$ に出

[2]GJ, FGJ に倣い，$\triangleleft$ (extends) には微妙に異なるふたつの用法があることに注意されたい．ひとつはクラス間の継承関係を宣言するためであり，もうひとつは，このようにクラス・メソッドの型変数の部分型制約を記述するためである．

```
class Pair<X extends Object, Y extends Object> extends Object{
  X fst;
  Y snd;
  <Z extends Object> Pair<X,Y> setfst(Z x){ return new Pair<Z,Y>(x, this.snd); }
}
class C extends Object{
  C clone(){ return new C(); }
}
class D extends Object{
  D clone(){ return new D(); }
}
```

図 2: FGJ$\vee$ のクラス定義例

現するクラス名は (Object を除き) すべて $\overline{\mathtt{L}}$ 内で定義されているものとする．

以降の例では，クラス集合には図 2 のクラスが含まれていると仮定する．

3.2 補助関数

次に，与えられたクラスの集合からフィールドやメソッドの型情報を問い合わせるための関数群を定義する．これらは型システムの定義で用いられる補助的な関数であるが，実質的に union 型に対するメソッド起動などの規則を決定する重要な役割を担っている．定義される関数とその意味は，以下の通りである．

$\mathit{fields}(\mathtt{C<}\overline{\mathtt{T}}\mathtt{>})$	インスタンス型 C<$\overline{\mathtt{T}}$> の持つ全フィールド
$\mathit{ftype}(\mathtt{f},\mathtt{N})$	非変数型 N におけるフィールド f の型
$\mathit{mtype}(\mathtt{m},\mathtt{N})$	非変数型 N におけるメソッド m の型

フィールド定義・型の問い合わせ関数: $\mathit{fields}(\mathtt{C<}\overline{\mathtt{S}}\mathtt{>})$ はインスタンス型 C<$\overline{\mathtt{T}}$> が持つフィールドの名前と型の列 $\overline{\mathtt{T}}$ $\overline{\mathtt{f}}$ を表わし，そのクラス C および先祖におけるフィールドの宣言をすべて集めたものになる．このとき，クラスの型変数 $\overline{\mathtt{X}}$ は型引数 $\overline{\mathtt{T}}$ で具体化される．形式的定義は付録に譲る．

関数 $\mathit{ftype}(\mathtt{f},\mathtt{N})$ はフィールドアクセス式 e.f の型付けで用いられ，非変数型 N のフィールド f の型を表す．定義は以下の規則の通りである．

$$\frac{\mathit{fields}(\mathtt{C<}\overline{\mathtt{S}}\mathtt{>}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\mathit{ftype}(\mathtt{f}_i,\mathtt{C<}\overline{\mathtt{S}}\mathtt{>}) = \mathtt{T}_i} \qquad \frac{\mathit{ftype}(\mathtt{f},\mathtt{N}) = \mathtt{T} \quad \mathit{ftype}(\mathtt{f},\mathtt{P}) = \mathtt{S}}{\mathit{ftype}(\mathtt{f},\mathtt{N}\vee\mathtt{P}) = \mathtt{T}\vee\mathtt{S}}$$

インスタンス型のフィールド型は先に定義した *fields* から導かれる．二番目の規則により union 型 N$\vee$P のフィールドの型は，それぞれのフィールドの型の union となる．これにより，N，P が同名のフィールド f を持つ場合，例えそれが別のクラスに由来するものであっても，アクセスが可能になる．

例 1 fields(Pair<C,Object>) = C fst, Object snd より ftype(fst, Pair<C,Object>) = C．同様に ftype(fst, Pair<D,Object>) = D．また，ftype(fst, Pair<C,Object>$\vee$Pair<D,Object>) = C$\vee$D．

メソッド型問い合わせ関数: $\mathit{mtype}(\mathtt{m},\mathtt{T})$ は型 T のメソッド m の型情報を返す．この型情報は，<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}_1\wedge\cdots\wedge\overline{\mathtt{N}}_n$>($\overline{\mathtt{T}}_1\wedge\cdots\wedge\overline{\mathtt{T}}_n$)$\to$T (ただし $n \geq 1$) という形で表され，

メソッドの型パラメータが $\overline{\mathtt{X}}$ であり，$\overline{\mathtt{X}}$ に渡される具体的な型はその上限 $\overline{\mathtt{N}}_i$ を全て満さなければならず，メソッド引数の型が $\overline{\mathtt{T}}_i$ すべての部分型である時に T 型の値を返す

ということを意味する． 上の表記で $\overline{\mathtt{X}}$ は束縛変数であり，有効範囲は $\overline{\mathtt{N}}_1, \ldots, \overline{\mathtt{N}}_n, \overline{\mathtt{T}}_1, \ldots, \overline{\mathtt{T}}_n$, T である．束縛変数の名前替えは適宜暗黙のうちに行えるものとする．定義は以下の規則で与えられる．

$$\frac{\texttt{class C<}\overline{\mathtt{X}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{N}}\texttt{>}\triangleleft\texttt{D<}\overline{\mathtt{V}}\texttt{> \{}\overline{\mathtt{S}}\ \overline{\mathtt{f}}\texttt{; }\overline{\mathtt{M}}\texttt{\} <}\overline{\mathtt{Y}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{P}}\texttt{> U m(}\overline{\mathtt{U}}\ \overline{\mathtt{x}}\texttt{)\{ return e; \}} \in \overline{\mathtt{M}}}{mtype(\mathtt{m}, \mathtt{C<}\overline{\mathtt{T}}\mathtt{>}) = [\overline{\mathtt{T}}/\overline{\mathtt{X}}](\texttt{<}\overline{\mathtt{Y}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{P}}\texttt{>(}\overline{\mathtt{U}}\texttt{)}\rightarrow\mathtt{U})}$$

$$\frac{\texttt{class C<}\overline{\mathtt{X}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{N}}\texttt{>}\triangleleft\texttt{D<}\overline{\mathtt{V}}\texttt{> \{}\overline{\mathtt{S}}\ \overline{\mathtt{f}}\texttt{; }\overline{\mathtt{M}}\texttt{\} m} \notin \overline{\mathtt{M}}}{mtype(\mathtt{m}, \mathtt{C<}\overline{\mathtt{T}}\mathtt{>}) = mtype(\mathtt{m}, \mathtt{D<}[\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{V}}\mathtt{>})}$$

$$\frac{\begin{array}{c} mtype(\mathtt{m}, \mathtt{N}) = \texttt{<}\overline{\mathtt{X}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{N}}_1\wedge\ldots\wedge\overline{\mathtt{N}}_n\texttt{>(}\overline{\mathtt{S}}_1\wedge\ldots\wedge\overline{\mathtt{S}}_n\texttt{)}\rightarrow\mathtt{S} \\ mtype(\mathtt{m}, \mathtt{P}) = \texttt{<}\overline{\mathtt{X}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{P}}_1\wedge\ldots\wedge\overline{\mathtt{P}}_k\texttt{>(}\overline{\mathtt{T}}_1\wedge\ldots\wedge\overline{\mathtt{T}}_k\texttt{)}\rightarrow\mathtt{T} \end{array}}{mtype(\mathtt{m}, \mathtt{N}\vee\mathtt{P}) = \texttt{<}\overline{\mathtt{X}}\texttt{}\triangleleft\texttt{}\overline{\mathtt{N}}_1\wedge\ldots\wedge\overline{\mathtt{N}}_n\wedge\overline{\mathtt{P}}_1\wedge\ldots\wedge\overline{\mathtt{P}}_k\texttt{>(}\overline{\mathtt{S}}_1\wedge\ldots\wedge\overline{\mathtt{S}}_n\wedge\overline{\mathtt{T}}_1\wedge\ldots\wedge\overline{\mathtt{T}}_k\texttt{)}\rightarrow\mathtt{S}\vee\mathtt{T}}$$

インスタンス型 $\mathtt{C<}\overline{\mathtt{T}}\mathtt{>}$ のメソッド型は m の定義が発見されるまで親クラスを辿っていき，発見された定義の型を (型変数を具体化して) 返す素直な定義である．union 型 N∨P に対するメソッド呼出しは，実行時に N，P のどちらのオブジェクトであっても呼出せなくてはならないため，最低限，メソッドが両方に存在し，その引数の数が一致していなければならない．型に関する最も保守的な条件としては，$mtype(\mathtt{m}, \mathtt{N})$ と，$mtype(\mathtt{m}, \mathtt{P})$ が一致していなくてはならないという条件が考えられるが，ここではできるだけ条件を緩めて許可するように定義している．即ち，(型) 引数の個数が一致していれば，$mtype(\mathtt{m}, \mathtt{N}\vee\mathtt{P})$ は定義され，$mtype(\mathtt{m}, \mathtt{N})$ と $mtype(\mathtt{m}, \mathtt{P})$ のもつ (型) 引数の条件をともにもつ (∧) メソッド型を返す．メソッドの返り値の型はフィールド呼び出しと同様に union 型となる．

例 2 $mtype(\mathtt{clone}, \mathtt{C}) = \mathtt{()}\rightarrow\mathtt{C}$, 同様に $mtype(\mathtt{clone}, \mathtt{D}) = \mathtt{()}\rightarrow\mathtt{D}$. これらより $mtype(\mathtt{clone}, \mathtt{C}\vee\mathtt{D}) = \mathtt{()}\rightarrow\mathtt{C}\vee\mathtt{D}$. (空の <> や空列を ∧ で繋げたものは省略している．) また,

$$mtype(\mathtt{setfst}, \mathtt{Pair<Object,C>}\vee\mathtt{Pair<Object,D>}) = \\ \mathtt{<Z\ \triangleleft Object\ \wedge\ Object>(Z\wedge Z)}\rightarrow\mathtt{Pair<Z,C>}\vee\mathtt{Pair<Z,D>}$$

である．

本稿では，メソッド呼出しに関する制限をなるべく緩くして体系を定義し，その緩い制限の元でも型健全性が成立することを示しているが，現実の言語としてはもう少し制限をきつくする必要があるかもしれないと考えている．特にここでモデル化されていないメソッドの overloading の解決と union 型の相互作用についてはなお検討を要すると考えている．

3.3 型システム

FGJ∨ の型システムは，以下に示す判断から構成される．

部分型判断	$\Delta \vdash \mathtt{S} <: \mathtt{T}$
型の正しさの判断	$\Delta \vdash \mathtt{T}$ ok
式の型判断	$\Delta;\Gamma \vdash \mathtt{e} : \mathtt{T}$
メソッドの型判断	$\mathtt{M}$ OK IN $\mathtt{C}<\overline{\mathtt{X}} \lhd \overline{\mathtt{N}}>$
クラスの型判断	$\mathtt{L}$ OK

ここで，Δ は**型制限**と呼ばれ，$\overline{\mathtt{X}}<:\overline{\mathtt{N}}$ の形の (空かもしれない) 列である．意味的には各 $\mathtt{X}_i$ が $\mathtt{N}_i$ の部分型であるという仮定を示し，形式的には型変数から型の上限への関数として扱われる．また，Γ を**型環境**と呼ぶ，$\mathtt{x}_1:\mathtt{T}_1, \ldots, \mathtt{x}_n:\mathtt{T}_n$ ($\overline{\mathtt{x}}:\overline{\mathtt{T}}$ と略記される) の形の列で，変数 $\mathtt{x}_i$ が $\mathtt{T}$ 型を持つという仮定を表す．形式的には変数から型への (部分) 関数として扱う．空の型制限/型環境を $\emptyset$ と書く．

以下では主要な判断である，部分型判断と式の型判断について詳しく説明する．

3.3.1 部分型規則

部分型判断 $\Delta \vdash \mathtt{S}<:\mathtt{T}$ は「型制限 Δ の下で $\mathtt{S}$ は $\mathtt{T}$ の部分型である」をいうことを意味する．部分型判断を導く規則は以下のように定義される．$[\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}$ は $[\mathtt{T}_1/\mathtt{X}_1, \ldots, \mathtt{T}_n/\mathtt{X}_n]\overline{\mathtt{S}}$ の略記であり，$\overline{\mathtt{S}}$ の型変数 $\overline{\mathtt{X}}$ を $\overline{\mathtt{T}}$ によって置き換える一般的な代入を表す．

$$\frac{\mathtt{class\ C}<\overline{\mathtt{X}} \lhd \overline{\mathtt{N}}> \lhd \mathtt{D}<\overline{\mathtt{S}}>\ \{\ldots\}}{\Delta \vdash \mathtt{C}<\overline{\mathtt{T}}>\ <:\ \mathtt{D}<[\overline{\mathtt{T}}/\overline{\mathtt{X}}]\overline{\mathtt{S}}>} \qquad \frac{\Delta \vdash \overline{\mathtt{S}}<:\overline{\mathtt{T}} \quad \Delta \vdash \overline{\mathtt{T}}<:\overline{\mathtt{S}}}{\Delta \vdash \mathtt{C}<\overline{\mathtt{S}}>\ <:\ \mathtt{C}<\overline{\mathtt{T}}>}$$

$$\Delta \vdash \mathtt{S} <: \mathtt{S} \vee \mathtt{T} \qquad \Delta \vdash \mathtt{T} <: \mathtt{S} \vee \mathtt{T} \qquad \frac{\Delta \vdash \mathtt{S}<:\mathtt{U} \quad \Delta \vdash \mathtt{T}<:\mathtt{U}}{\Delta \vdash \mathtt{S} \vee \mathtt{T} <: \mathtt{U}}$$

この他に，部分型関係が反射的推移的であることを示す規則や型変数が型制限中の上限と部分型関係にあることを導く規則があるが，ここでは省略する．最初の規則は，部分型関係が継承関係から (型変数を具体化することで) 導かれることを示す．また，2 段目の 3 つの規則によって $\mathtt{S}\vee\mathtt{T}$ は $\mathtt{S}$ と $\mathtt{T}$ の上限となる．これにより部分型関係は反対称的ではなくなることに注意したい．例えば $\mathtt{class\ C} \lhd \mathtt{D\{\ldots\}}$ であるとき $\emptyset \vdash \mathtt{C} \vee \mathtt{D} <: \mathtt{D}$ かつ $\emptyset \vdash \mathtt{D} <: \mathtt{C} \vee \mathtt{D}$ である．このようにお互いに部分型関係が成り立つとき二つの型は**互換**であるという．最後に 1 段目の右側の規則はクラスを互換な型で具体化しても互換となることを表す．($\mathtt{T}_1 <: \mathtt{T}_2$ ならば $\mathtt{C}<\mathtt{T}_1> <: \mathtt{C}<\mathtt{T}_2>$ といった，いわゆる covariant な部分型規則ではないことに注意されたい．covariant な部分型規則を採り入れた型システムは不健全になってしまう．) 以下，$\Delta \vdash \mathtt{S}_1 <: \mathtt{T}_1, \ldots, \Delta \vdash \mathtt{S}_n <: \mathtt{T}_n$ を $\Delta \vdash \overline{\mathtt{S}} <: \overline{\mathtt{T}}$ と略す．

例 3 仮定されているクラス集合の元で，以下の部分型判断が導出できる．

$$\emptyset \vdash \mathtt{C} <: \mathtt{C} \vee \mathtt{D} \qquad \emptyset \vdash \mathtt{D} <: \mathtt{C} \vee \mathtt{D} \qquad \emptyset \vdash \mathtt{C} \vee \mathtt{D} <: \mathtt{Object}$$

最後の判断は $\emptyset \vdash \mathtt{C} <: \mathtt{Object}$，$\emptyset \vdash \mathtt{D} <: \mathtt{Object}$ より導かれる．

3.3.2 型付け規則

式に対する型付け判断 $\Delta;\Gamma \vdash \mathtt{e} : \mathtt{T}$ は，型制限 Δ と 型環境 Γ の下で $\mathtt{e}$ が型 $\mathtt{T}$ を持つことを意味する．以下で，$\Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{T}}$ は $\Delta;\Gamma \vdash \mathtt{e}_1 : \mathtt{T}_1, \ldots, \Delta;\Gamma \vdash \mathtt{e}_n : \mathtt{T}_n$ の略記である．

$bound_\Delta(\mathtt{T})$ を型制限 Δ のもとでの型 $\mathtt{T}$ の非変数型であるような上限を表す (定義は付録参照) とし，型付け規則は以下のように定義される．

$$\Delta;\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x})$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \mathit{ftype}(\mathtt{f}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{T}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{T}}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \texttt{<}\overline{\mathtt{X}} \triangleleft \overline{\mathtt{N}}_1 \wedge \ldots \wedge \overline{\mathtt{N}}_n\texttt{>}(\overline{\mathtt{U}}_1 \wedge \ldots \wedge \overline{\mathtt{U}}_n)\rightarrow \mathtt{U}_0 \\ \Delta \vdash \overline{\mathtt{V}} \ \mathrm{ok} \quad \Delta \vdash \overline{\mathtt{V}} \texttt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}_1 \quad \cdots \quad \Delta \vdash \overline{\mathtt{V}} \texttt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}_n \\ \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{T}} \quad \Delta \vdash \overline{\mathtt{T}} \texttt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{U}}_1 \quad \cdots \quad \Delta \vdash \overline{\mathtt{T}} \texttt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{U}}_n\end{array}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}(\overline{\mathtt{e}}) : [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\mathtt{U}_0}$$

$$\frac{\Delta \vdash \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>} \ \mathrm{ok} \quad \mathit{fields}(\mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}) = \overline{\mathtt{T}} \ \overline{\mathtt{f}} \quad \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \quad \Delta \vdash \overline{\mathtt{S}} \texttt{<:} \overline{\mathtt{T}}}{\Delta;\Gamma \vdash \texttt{new } \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}(\overline{\mathtt{e}}) : \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}}$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \Delta \vdash \mathtt{T}_0 \texttt{<:} \mathtt{T}_1 \vee \mathtt{T}_2 \quad \Delta \vdash \mathtt{T}_1 \vee \mathtt{T}_2 \ \mathrm{ok} \quad \Delta;\Gamma, \mathtt{x}\texttt{:}\mathtt{T}_1 \vdash \mathtt{e}_1 \texttt{:} \mathtt{S}_1 \quad \Delta;\Gamma, \mathtt{y}\texttt{:}\mathtt{T}_2 \vdash \mathtt{e}_2 \texttt{:} \mathtt{S}_2}{\Delta;\Gamma \vdash \texttt{case } \mathtt{e}_0 \texttt{ of } (\mathtt{T}_1 \ \mathtt{x}) \ \mathtt{e}_1 \texttt{ | } (\mathtt{T}_2 \ \mathtt{y}) \ \mathtt{e}_2 : \mathtt{S}_1 \vee \mathtt{S}_2}$$

フィールドアクセス・インスタンス生成は補助関数 *ftype*, *fields* を用いてフィールドの型情報を問い合わせる．さらに，インスタンス生成では，実引数の型とフィールドの型の整合性を検査する．メソッド呼び出しも同様に補助関数 *mtype* でメソッドの型情報を問い合わせる．この時，型引数 $\overline{\mathtt{V}}$ が，対応する型変数に課された全ての上限を満たすことや，実引数 $\overline{\mathtt{e}}$ の型がパラメータの型の部分型であることを検査する．case 式の規則における $\mathtt{T}_1$ と $\mathtt{T}_2$ は，場合わけでマッチするオブジェクトの型を示しているため，対象となる $\mathtt{e}_0$ の型 $\mathtt{T}_0$ はそれらの和の部分型であることが必要となる．また，全体の型はどちらかの分岐が実行されるため，各分岐の型の和となる．

例 4 X<:C∨D; x:X ⊢ case x of (C y) y.clone() | (D z) z.clone() : C∨D が導出できる．

例 5 式 e1 を new Pair<C∨D,Object>(new C(), new Object()).setfst<C∨D>(new D()) とする．これに対し，$\emptyset;\emptyset \vdash \mathtt{e}_1 : \mathtt{Pair}\texttt{<}\mathtt{C}\vee\mathtt{D}\texttt{,}\mathtt{Object}\texttt{>}$ が導出できる．このことから，$\emptyset;\emptyset \vdash \mathtt{e}_1.\mathtt{fst} : \mathtt{C}\vee\mathtt{D}$．さらに $\mathit{mtype}(\mathtt{clone}, \mathtt{C}\vee\mathtt{D}) = ()\rightarrow\mathtt{C}\vee\mathtt{D}$ より $\emptyset;\emptyset \vdash \mathtt{e}_1.\mathtt{fst.clone}() : \mathtt{C}\vee\mathtt{D}$ が導出できる．

例 6 式 e2

```
e2 = case new Pair<C∨D,Object>(new C(), new Object()).fst of
       (C x) x.clone() | (D y) new Pair<D,D>(y,y)
```

について，$\emptyset;\emptyset \vdash \mathtt{e}_2 : \mathtt{C}\vee\mathtt{Pair}\texttt{<}\mathtt{D}\texttt{,}\mathtt{D}\texttt{>}$ が導出できる．

メソッド定義の型付け判断 M OK IN C<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> は，メソッド M がクラス class C<$\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}$> ... 中に現われるものとして正しいことを示す．導出規則のひとつは以下のものである．

$$\frac{\begin{array}{c}\Delta = \overline{\mathtt{X}}\texttt{<:}\overline{\mathtt{N}}\texttt{,}\overline{\mathtt{Y}}\texttt{<:}\overline{\mathtt{P}} \quad \Delta \vdash \overline{\mathtt{T}}, \mathtt{T}, \overline{\mathtt{P}} \ \mathrm{ok} \\ \Delta;\overline{\mathtt{x}}\texttt{:}\overline{\mathtt{T}}\texttt{, this} : \mathtt{C}\texttt{<}\overline{\mathtt{X}}\texttt{>} \vdash \mathtt{e}_0\texttt{:}\mathtt{S} \quad \Delta \vdash \mathtt{S}\texttt{<:}\mathtt{T} \\ \texttt{class } \mathtt{C}\texttt{<}\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}\texttt{>}\triangleleft\mathtt{D}\texttt{<}\overline{\mathtt{V}}\texttt{>} \texttt{ \{ ... \}} \\ \mathit{mtype}(\mathtt{m}, \mathtt{D}\texttt{<}\overline{\mathtt{V}}\texttt{>}) = \texttt{<}\overline{\mathtt{Y}}\triangleleft\overline{\mathtt{P}}\texttt{>}(\overline{\mathtt{T}})\rightarrow\mathtt{T}' \quad \overline{\mathtt{Y}}\texttt{<:}\overline{\mathtt{P}} \vdash \mathtt{T}\texttt{<:}\mathtt{T}'\end{array}}{\texttt{<}\overline{\mathtt{Y}}\triangleleft\overline{\mathtt{P}}\texttt{> T m(}\overline{\mathtt{T}}\ \overline{\mathtt{x}}\texttt{)\{ return } \mathtt{e}_0\texttt{; \} OK IN C<}\overline{\mathtt{X}}\triangleleft\overline{\mathtt{N}}\texttt{>}}$$

これは親クラスのメソッドを上書き再定義している場合の規則である．メソッド宣言中に出現する型が正しいこと，メソッド本体が型付けされること，メソッドの型が親クラスのものと整合性がとれていることを検査する．メソッド本体の型付けのための型環境には，宣言されたパラメータの型に加え，this の型情報が含まれている．

クラスに対する型付け規則は，宣言中に現れる型がすべて正しく，メソッドが全て型付けされる場合に型付けされるという単純なものであるので，ここでは省略する．プログラム $(\overline{\mathtt{L}}, \mathtt{e})$ は，すべてのクラス $\mathtt{L}_i$ が正しく型付けされ，$\emptyset;\emptyset \vdash \mathtt{e} : \mathtt{T}$ が，ある T に対して成り立つとき，プログラムが型付けされている，と定義する．

3.4 操作的意味

FGJ$\vee$ の操作的意味は簡約関係を使って定義する．簡約関係は $\mathtt{e} \longrightarrow \mathtt{e}'$ と表され，e が $\mathtt{e}'$ に 1 ステップで簡約されることを意味する．この関係は下に示す基本的な簡約規則を一つの部分式に適用することによって導出される．

(メソッド呼び出しのための) 簡約規則中，使用される補助関数 $mbody(\mathtt{m<\overline{V}>}, \mathtt{C<\overline{T}>})$ はインスタンス型 C<$\overline{\mathtt{T}}$> におけるメソッド m を型引数 $\overline{\mathtt{V}}$ で呼び出した時の仮引数と本体の組 $\overline{\mathtt{x}}.\mathtt{e}$ を表す．(形式的定義は省略するが $mtype$ と同様に定義される．) $[\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{e}'/\mathtt{this}]\mathtt{e}$ は式 e 中の $\overline{\mathtt{x}}$, this に $\overline{\mathtt{d}}$, $\mathtt{e}'$ を代入した式を示す．式の代入の定義は省略するが，束縛変数を考慮して定義される標準的なものである．

以下は式の簡約規則であり，これらを部分式に適用することによって簡約関係が定義される．

$$\frac{fields(\mathtt{C<\overline{S}>}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\mathtt{(new\ C<\overline{S}>(\overline{e})).f}_i \longrightarrow \mathtt{e}_i}$$

$$\frac{mbody(\mathtt{m<\overline{V}>}, \mathtt{C<\overline{T}>}) = \overline{\mathtt{x}}.\mathtt{e}_0}{\mathtt{(new\ C<\overline{T}>(\overline{e})).m<\overline{V}>(\overline{d})} \longrightarrow [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new\ C<\overline{T}>(\overline{e})}/\mathtt{this}]\mathtt{e}_0}$$

$$\frac{\emptyset \vdash \mathtt{C<\overline{T}><:N}_1}{\mathtt{case\ new\ C<\overline{T}>(\overline{e})\ of\ (N}_1\ \mathtt{x)e}_1\mathtt{|(N}_2\ \mathtt{y)e}_2 \longrightarrow [\mathtt{new\ C<\overline{T}>(\overline{e})}/\mathtt{x}]\mathtt{e}_1}$$

$$\frac{\emptyset \vdash \mathtt{C<\overline{T}>} \not<: \mathtt{N}_1 \quad \emptyset \vdash \mathtt{C<\overline{T}><:N}_2}{\mathtt{case\ new\ C<\overline{T}>(\overline{e})\ of\ (N}_1\ \mathtt{x)e}_1\mathtt{|(N}_2\ \mathtt{y)e}_2 \longrightarrow [\mathtt{new\ C<\overline{T}>(\overline{e})}/\mathtt{y}]\mathtt{e}_2}$$

フィールド，メソッドの簡約は補助関数によってクラス定義の情報をもとに定義される．case の簡約規則によって，場合分けの対象の式が左側の条件 ($\mathtt{N}_1$ の部分型のインスタンスであること) を満たす場合は左側の計算，そうではなく右側の条件を満たす場合は右側の計算を行う．

例 7 例 5 の式 e1 は以下のように簡約される．

```
e1  →  [new Pair<C∨D,Object>(...)/this,new D()/x]new Pair<C∨D,Object>(x,this.snd)
    =  new Pair<C∨D,Object>(new D(),new Pair<C∨D,Object>(...).snd)
    →  new Pair<C∨D,Object>(new D(),new Object())
```

また，例 6 の式 e2 は以下のように簡約される．

```
e2  →  case new C() of (C x) x.clone() | (D y) new Pair<D,D>(y,y)
    →  new C().clone()     (= [new C()/x]x.clone())
    →  new C()     (= [new C()/this]new C())
```

4 FGJ∨ の性質

本節では，FGJ∨ の型付けによる安全性を保証する型健全性について述べる．また FGJ∨ におけるプログラムの計算過程は一意的ではないため計算結果が一意的であることは自明ではない．計算結果の一意性についての定理もあわせて述べる．

型健全性に用いられる主定理は簡約前後で型付けが保存されるという Subject Reduction，および，型付けされる式は値で無い限り簡約することができることを保証する Progress であり，型健全性はそれらから導かれる．

定理 1 (Subject Reduction) $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}$ かつ $\mathtt{e} \longrightarrow \mathtt{e}'$ ならば $\Delta;\Gamma \vdash \mathtt{e}:\mathtt{T}'$ かつ $\Delta \vdash \mathtt{T}'<:\mathtt{T}$ なる $\mathtt{T}'$ が存在する．

(証明概略) $\mathtt{e} \longrightarrow \mathtt{e}'$ の導出に関する帰納法．証明に必要となる主な補題は次のようなものである．

メソッド呼出しの場合に，以下のメソッドの型付け規則からメソッド本体を表す式の型付けを抽出するための補題といわゆる代入補題が使われる．

補題 1 $mtype(\mathtt{m},\mathtt{C<\overline{T}>}) = \mathtt{<\overline{Y} \lhd \overline{P}>\overline{U} \to U}$ かつ $mbody(\mathtt{m<\overline{V}>},\mathtt{C<\overline{T}>}) = \mathtt{\overline{x}.e_0}$ かつ $\Delta \vdash \mathtt{C<\overline{T}>,\overline{V}}$ かつ $\Delta \vdash \mathtt{\overline{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\overline{\mathtt{P}}$ ならば，$\Delta \vdash \mathtt{C<\overline{T}>} <: \mathtt{D<\overline{S}>}$ かつ $\Delta \vdash \mathtt{D<\overline{S}>ok}$ かつ $\Delta \vdash \mathtt{S} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{U}$ かつ $\Delta;\mathtt{\overline{x}}:[\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U},this:D<\overline{S}>} \vdash \mathtt{e_0}:\mathtt{S}$ なる $\mathtt{D<\overline{S}>,S}$ が存在する．

補題 2 $\Delta;\Gamma,\mathtt{\overline{x}}:\mathtt{\overline{T}} \vdash \mathtt{e}:\mathtt{T}$ かつ $\Delta;\Gamma \vdash \mathtt{\overline{d}}:\mathtt{\overline{S}}$ かつ $\Delta \vdash \mathtt{\overline{S}} <: \mathtt{\overline{T}}$ ならば，$\Delta;\Gamma \vdash [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}:\mathtt{S}$ かつ $\Delta \vdash \mathtt{S} <: \mathtt{T}$ なる S が存在する．

次のふたつの補題はフィールドアクセスや，メソッド呼び出し式で操作対象の式が簡約される際 ($\mathtt{e} \longrightarrow \mathtt{e}'$ から $\mathtt{e.f} \longrightarrow \mathtt{e}'\mathtt{.f}$ が導出されている場合など) に用いられる．

補題 3 $ftype(\mathtt{f},\mathtt{N}) = \mathtt{T}$ かつ $\Delta \vdash \mathtt{P} <: \mathtt{N}$ ならば $ftype(\mathtt{f},\mathtt{P}) = \mathtt{U}$ かつ $\Delta \vdash \mathtt{U} <: \mathtt{T}$ なる U が存在する．

補題 4 $mtype(\mathtt{m},P) = \mathtt{<\overline{Y} \lhd \overline{P}_1 \wedge \ldots \wedge \overline{P}_n>(\overline{U}_1 \wedge \ldots \wedge \overline{U}_n) \to U_0}$ かつ $\Delta \vdash \mathtt{\overline{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{P}}_1 \cdots \Delta \vdash \mathtt{\overline{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{P}}_n$ かつ $\Delta \vdash \mathtt{\overline{T}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U}}_1 \cdots \Delta \vdash \mathtt{\overline{T}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U}}_n$ かつ $\Delta \vdash \mathtt{P'} <: \mathtt{P}$ ならば $\exists mtype(\mathtt{m},P') = \mathtt{<\overline{Y} \lhd \overline{P'}_1 \wedge \ldots \wedge \overline{P'}_k>(\overline{U'}_1 \wedge \ldots \wedge \overline{U'}_k) \to U'_0}$ かつ $\Delta \vdash \mathtt{\overline{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{P'}}_1 \cdots \Delta \vdash \mathtt{\overline{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{P'}}_k$ かつ $\Delta \vdash \mathtt{\overline{T}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U'}}_1 \cdots \Delta \vdash \mathtt{\overline{T}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U'}}_k$ かつ $\Delta \vdash [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{\overline{U'}}_0 <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{U}_0$.

定理 2 (Progress) $\emptyset;\emptyset \vdash \mathtt{e}:\mathtt{T}$ かつ e が値でないならば，$\mathtt{e} \longrightarrow \mathtt{e}'$ なる $\mathtt{e}'$ が存在する．

(証明概略) e が値でない閉式であるため，その部分式には次のいずれかの形のものが存在し，簡約するため条件 (i)–(iii) を満たすことを示す．いずれの条件も型付けされていることから簡単に導くことができる．

(i) $\mathtt{new\ C<\overline{U}>(\overline{e}).f}$ のとき，$fields(\mathtt{C<\overline{U}>}) = \mathtt{\overline{T}\ \overline{f}}$ かつ $\mathtt{f} \in \mathtt{\overline{f}}$ なる $\mathtt{\overline{T}}$，$\mathtt{\overline{f}}$ が存在する．

(ii) `new C<`$\overline{\mathtt{U}}$`>(`$\overline{\mathtt{e}}$`).m<`$\overline{\mathtt{V}}$`>(`$\overline{\mathtt{d}}$`)` のとき，$mbody(\mathtt{m}\texttt{<}\overline{\mathtt{V}}\texttt{>}, \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}) = \overline{\mathtt{x}}.\mathtt{e}_0$ かつ $\#(\overline{\mathtt{x}}) = \#(\overline{\mathtt{d}})$ なる $\overline{\mathtt{x}}$，$\mathtt{e}_0$ が存在する．

(iii) `case new C<`$\overline{\mathtt{U}}$`>(`$\overline{\mathtt{e}}$`) of (`$\mathtt{T}_1$ $\mathtt{x}_1$`)`$\mathtt{d}_1$ `|` `(`$\mathtt{T}_2$ $\mathtt{x}_2$`)`$\mathtt{d}_2$ のとき，$\emptyset \vdash \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}\texttt{<:}\mathtt{T}_1$ または $\emptyset \vdash \mathtt{C}\texttt{<}\overline{\mathtt{U}}\texttt{>}\texttt{<:}\mathtt{T}_2$

Progress は case 式の場合分けの完全性，Union 型からの同名メンバ直接呼び出しの際に，呼び出しが正しく行われることなどを保証する．

次の型健全性は Subject Reduction および Progress よりすぐに導かれる．

定理 3 (Type Soundness) $\emptyset;\emptyset \vdash \mathtt{e}:\mathtt{T}$ かつ $\mathtt{e} \longrightarrow^* \mathtt{e}'$ かつ e が *normal form* ならば，$\mathtt{e}'$ は値であり，$\emptyset;\emptyset \vdash \mathtt{e}':\mathtt{S}$ かつ $\emptyset \vdash \mathtt{S}\texttt{<:}\mathtt{T}$ なる S が存在する．

次の定理は，計算過程によらず結果は一意的であることを主張する．

定理 4 (計算結果の一意性) $\emptyset;\emptyset \vdash \mathtt{e}:\mathtt{T}$ かつ $\mathtt{e} \longrightarrow^* \mathtt{v}_1$ かつ $\mathtt{e} \longrightarrow^* \mathtt{v}_2$ ならば $\mathtt{v}_1 = \mathtt{v}_2$

この証明は並行簡約の技法を使って示される合流性から得られる．

5 関連研究

プログラミング言語において，和集合的なデータを表現するための機構としては，ML [15] の datatype のように，その型の値を構成する際にタ明示的なグ付けを伴なうものと，C 言語の共用体のようにタグ付けを伴なわないものに分けられる．前者は，タグによる場合分け処理が言語に備わっているが，後者は (他にタグに相当する情報を付加するなどの処理を明示的に行なわない限り) 言語機構としては場合分け処理が用意されていない．本研究で提案する FGJ∨ では，オブジェクトに付加されているクラス情報をタグと見なすことで，明示的なタグ付けをなくし，かつ，場合分けが可能な機構を実現している．言い換えれば，タグ付けはインスタンス生成時に暗黙のうちに行われていると考えられる．

ML の datatype では，ひとつのタグを用いて構成できるのはタグと同時に宣言された型の値だけであるが，FGJ∨ では任意の型同士の和を宣言なしに自由に使うことができ，かつ，ひとつのタグ (クラス) は様々な union 型の値とみなすことができる．逆に，我々の union 型では，同じ型の和を構成しても case で区別できないので意味がない (実際 T と T∨T は互換である) が，datatype では，同じ型に違ったタグを用意することで case で区別することができる．Objective Caml の polymorphic variant [10] は，ひとつのタグから複数の union 型の値を構成できるという意味でより我々の union 型に近い．

Java に generics を加えた拡張のひとつであり，GJ の元となった Pizza 言語 [18] では，ML の datatype と同様な機構が提案されていた．しかし，union 型同士の部分型などを継承に基いた自然な形で導入している FGJ∨ とは違い，オブジェクト指向言語の重要な要素である継承や部分型との融合について詳しく議論されてきていない．

C の共用体のように値の構成時のタグ付けがなく，場合分けができないような union 型の理論は型付き λ 計算で研究 [2] されてきている．また，近年 XML のような半構造データ処理のための言語機構として union 型が注目されている [5, 11]．FGJ∨ での，C, D が持つ同名のメンバに C∨D から直接アクセスできる機能は，それらの型システムにおける，union 型のレコード型に対する分配則にヒントを得たものである．Xtatic 言語 [9] は C# に XML 処理のための言語機構を導入した拡張で，union 型を含む正則表現型 [11] が備わっている．ただし FGJ∨ とは違い，XML を表す型とオブジェクト型は区別されているため，任意のオブジェクト型の和を考えられるわけではない．

汎用クラスの機構はオブジェクト指向言語では古くから研究されてきており [14, 16, 7, 17, 1] Java の普及により，その拡張として様々な提案 [3, 16, 7, 17, 1] がなされてきたが，union 型を併せ持つオブジェクト指向言語は (我々の知る限り) 存在していない．汎用クラスの概念の型理論的な基盤であるパラメトリック多相型と union 型との組合せは型付き λ 計算で研究されているが [19], その型理論の性質 (特に型検査/部分型検査の決定可能性・アルゴリズム) はまだ明らかにされていない．

6 おわりに

本稿では，クラスに基づくオブジェクト指向言語の，汎用クラスと union 型による拡張を議論した．generics だけでなく，動的型検査 (Java の `instanceOf`) とダウンキャストを統合した `case` 構文や，union 型で組み合わされたクラスの同名メンバへの直接アクセス，またそれらを併用することにより，潜在的危険のあるダウンキャストを使うことなく，柔軟なプログラム記述が可能になると考えている．また，これらの機構の安全性を確認するために，汎用クラス・union 型を導入した核言語 FGJ$\vee$ を形式化し，文法・型システム・操作的意味論を与え，型システムの健全性・計算結果の一意性を証明した．

今後の課題としては，まず，union 型の実装が挙げられる．素朴な実装としては，GJ が Java への変換で実装されているのと同様に，`case` 構文などを動的型検査と型変換で表現する方法が考えられるが，これではコンパイル後のコードの安全性は保証されないため，仮想機械の改造を含めて方向性を検討中である．

また，FGJ$\vee$ で扱っていない言語機構との組み合わせについてはさらに検討を要する．これまでの検討を通じて，メソッドのオーバーローディング機構との相性があまり良くないことが判明しており，今後の重要な課題のひとつである．

最近，汎用クラス上と部分型機構を改良するために variance に基づく部分型機構 [13] が提案されており，Java にも導入される予定である [20]．これは，同じ汎用クラスの異なる具体化に対する部分型を実現するための仕組みである．このような部分型の取り扱いは注意が必要であることが知られている．例えば `String` が `Object` の部分型であるからといって， 素朴に `List<String>` を `List<Object>` の部分型であると見なすのは一般的に安全ではない．この研究ではメソッドにアクセス制限を設けることで危険性を回避している．具体的には，アクセス制限はない従来と同じ型である `List<T>` と，リストへの書きこみのためのメソッド起動を禁止した `List<+T>` という型を導入し，`List<+T>` にのみ `List<S> <: List<+T>` (ただし `S<:T`) という部分型を認めるものである．この機構と union 型を組み合わせると，単一の要素のみからなるリストをデータが混在するリストであると見なせるようになると考えられる．例えば，文字列リスト型 `List<String>` と整数リスト型 `List<Integer>` は各々，(読み出しのみ可能な) `List<+(String∨Integer)>` の部分型として扱える．(FGJ$\vee$ では，これらは `List<String>∨List<Integer>` の部分型ではあるが，`List<String∨Integer>` の部分型ではない．) このような方向の言語の拡張により，generics と union 型を併せ持つことの意義がより一層増すと考えられる．

参考文献

[1] Ole Agesen, Stephen N. Freund, and John C. Mitchell. Adding type parameterization to the Java language. In *Proceedings of the OOPSLA'97*, pp. 49–65, Atlanta, GA, October 1997.

[2] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Intersection and union types: Syntax and semantics. *Information and Computation*, Vol. 119, No. 2, pp. 202–230, June 1995.

[3] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the Java programming language. In *Proc. of OOPSLA'98*, pp. 183–200, October 1998.

[4] Kim B. Bruce, Angela Schuett, and Robert van Gent. PolyTOIL: A type-safe polymorphic object-oriented language. In W. Olthoff, editor, *Proceedings of the ECOOP'95*, Vol. 952 of *LNCS*, pp. 27–51. Springer-Verlag, August 1995.

[5] Peter Buneman and Benjamin Pierce. Union types for semistructured data. In *Proc. of the International Database Programming Languages Workshop*, September 1999.

[6] Peter Canning, William Cook, Walter Hill, Walter Olthoff, and John Mitchell. F-bounded quantification for object-oriented programming. In *Fourth International Conference on Functional Programming Languages and Computer Architecture*, pp. 273–280, September 1989.

[7] Robert Cartwright and Guy L. Steele Jr. Compatible genericity with run-time types for the Java programming language. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'98)*, pp. 201–215, Vancouver, BC, October 1998.

[8] Mark Day, Robert Gruber, Barbara Liskov, and Andrew C. Myers. Subtypes vs. where clauses: Constraining parametric polymorphism. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 156–168, 1995.

[9] Vladimir Gapeyev and Benjamin Pierce. Regular object types. In *Proc. of ECOOP2003*, Vol. 2743 of *Springer LNCS*, pp. 151–175, July 2003.

[10] Jacques Garrigue. Programming with polymorphic variants. In *Proceedings of the ML Workshop*, September 1998.

[11] Haruo Hosoya, Jérôme Voullion, and Benjamin Pierce. Regular expression types for XML. In *Proc. of ACM ICFP*, pp. 11–22, September 2000.

[12] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. *ACM TOPLAS*, Vol. 23, No. 3, pp. 396–450, May 2001.

[13] Atsushi Igarashi and Mirko Viroli. On variance-based subtyping for parametric types. In *Proceedings of the ECOOP2002*, Vol. 2374 of *LNCS*, pp. 441–469, Málaga, Spain, June 2002. Springer-Verlag.

[14] Bertrand Meyer. Genericity versus inheritance. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 391–405, 1986.

[15] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML*. The MIT Press, revised edition, 1997.

[16] Andrew C. Myers, Joseph A. Bank, and Barbara Liskov. Parameterized types for Java. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, pp. 132–145, Paris, France, January 1997.

[17] Martin Odersky and Philip Wadler. Pizza into Java: Translating theory into practice. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 146–159, Paris, France, January 1997. ACM Press.

[18] Martin Odersky and Philip Wadler. Pizza into Java: Translating theory into practice. In *Proceedings of the POPL'97*, pp. 148–159, 1997.

[19] Benjamin C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1991.

[20] Mads Torgersen, Christian Plesner Hansen, Peter von der Ahé, Erik Ernst, Gilad Bracha, and Neal Gafter. Adding wildcards to the Java programming language. In *ACM Symposium on Applied Computing (OOPS Track)*, March 2004. To appear.

[21] 柳楽秀士. Generics・union 型を導入したオブジェクト指向計算体系. Master's thesis, 京都大学大学院情報学研究科知能情報学専攻, 2004.

[22] 柳楽秀士, 五十嵐淳. Union 型を導入したオブジェクト指向計算体系. 日本ソフトウェア科学会第 20 回大会論文集, September 2003.

A FGJ∨ の定義

Syntax:

N	::=	C<$\overline{\text{T}}$> | N∨N	非変数型
T	::=	X | C<$\overline{\text{T}}$> | T∨T	型
L	::=	class C<$\overline{\text{X}}$ ◁ $\overline{\text{N}}$> ◁ C<$\overline{\text{T}}$> {$\overline{\text{T}}$ $\overline{\text{f}}$; $\overline{\text{M}}$}	クラス定義
M	::=	<$\overline{\text{X}}$ ◁ $\overline{\text{N}}$> T m($\overline{\text{T}}$ $\overline{\text{x}}$){ return e; }	メソッド定義
e	::=	x | this | e.f | e.m<$\overline{\text{T}}$>($\overline{\text{e}}$) | new C<T>($\overline{\text{e}}$)	
	|	(case e of (T x) e | (T x) e)	式
v	::=	new C<$\overline{\text{T}}$>($\overline{\text{v}}$)	値

Subtyping:

$$\Delta \vdash \text{T} <: \text{T} \qquad \text{(S-Refl)}$$

$$\frac{\Delta \vdash \text{S} <: \text{T} \quad \Delta \vdash \text{T} <: \text{U}}{\Delta \vdash \text{S} <: \text{U}} \qquad \text{(S-Trans)}$$

$$\frac{\text{class C}\langle\overline{\text{X}} \lhd \overline{\text{N}}\rangle \lhd \text{D}\langle\overline{\text{S}}\rangle \ \{\ldots\}}{\Delta \vdash \text{C}\langle\overline{\text{T}}\rangle \ <: \ \text{D}\langle[\overline{\text{T}}/\overline{\text{X}}]\overline{\text{S}}\rangle} \qquad \text{(S-Class)}$$

$$\Delta \vdash \text{X} <: \Delta(\text{X}) \qquad \text{(S-Var)}$$

$$\frac{\Delta \vdash \overline{\text{S}} <: \overline{\text{T}} \quad \Delta \vdash \overline{\text{T}} <: \overline{\text{S}}}{\Delta \vdash \text{C}\langle\overline{\text{S}}\rangle \ <: \ \text{C}\langle\overline{\text{T}}\rangle} \qquad \text{(S-Comp)}$$

$$\Delta \vdash \text{S} <: \text{S} \vee \text{T} \qquad \text{(S-UniL)}$$

$$\Delta \vdash \text{T} <: \text{S} \vee \text{T} \qquad \text{(S-UniR)}$$

$$\frac{\Delta \vdash \text{S} <: \text{U} \quad \Delta \vdash \text{T} <: \text{U}}{\Delta \vdash \text{S} \vee \text{T} <: \text{U}} \qquad \text{(S-Bound)}$$

Well-formed types:

$$\Delta \vdash \texttt{Object}\ \text{ok} \qquad \text{(WF-Object)}$$

$$\Delta, \text{X} <: \text{N} \vdash \text{X}\ \text{ok} \qquad \text{(WF-Var)}$$

$$\frac{\text{class C}\langle\overline{\text{X}} \lhd \overline{\text{N}}\rangle \lhd \text{D}\langle\overline{\text{S}}\rangle\{\ldots\} \quad \Delta \vdash \overline{\text{T}}\ \text{ok} \quad \Delta \vdash \overline{\text{T}} <: [\overline{\text{T}}/\overline{\text{X}}]\overline{\text{N}}}{\Delta \vdash \text{C}\langle\overline{\text{T}}\rangle\ \text{ok}} \qquad \text{(WF-Class)}$$

$$\frac{\Delta \vdash \text{S}\ \text{ok} \quad \Delta \vdash \text{T}\ \text{ok}}{\Delta \vdash \text{S} \vee \text{T}\ \text{ok}} \qquad \text{(WF-Union)}$$

Field lookup:

$$\mathit{fields}(\texttt{Object}) = \bullet \qquad \text{(F-Object)}$$

$$\frac{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{V}}\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \quad \mathit{fields}(\texttt{D<}[\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{V}}\texttt{>}) = \overline{\texttt{U}}\ \overline{\texttt{g}}}{\mathit{fields}(\texttt{C<}\overline{\texttt{T}}\texttt{>}) = \overline{\texttt{U}}\ \overline{\texttt{g}}, [\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{S}}\ \overline{\texttt{f}}} \qquad \text{(F-Field)}$$

Field type lookup:

$$\frac{\mathit{fields}(\texttt{C<}\overline{\texttt{S}}\texttt{>}) = \overline{\texttt{T}}\ \overline{\texttt{f}}}{\mathit{ftype}(\texttt{f}_i, \texttt{C<}\overline{\texttt{S}}\texttt{>}) = \texttt{T}_i} \qquad \text{(FT-Field)}$$

$$\frac{\mathit{ftype}(\texttt{f}, \texttt{N}) = \texttt{T} \quad \mathit{ftype}(\texttt{f}, \texttt{P}) = \texttt{S}}{\mathit{ftype}(\texttt{f}, \texttt{N}\vee\texttt{P}) = \texttt{T}\vee\texttt{S}} \qquad \text{(FT-Union)}$$

Method type lookup:

$$\frac{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{V}}\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \quad \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\ \texttt{U m(}\overline{\texttt{U}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = [\overline{\texttt{T}}/\overline{\texttt{X}}](\texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>(}\overline{\texttt{U}}\texttt{)}\rightarrow\texttt{U})} \qquad \text{(MT-Class)}$$

$$\frac{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{V}}\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \quad \texttt{m} \notin \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = \mathit{mtype}(\texttt{m}, \texttt{D<}[\overline{\texttt{T}}/\overline{\texttt{X}}]\overline{\texttt{V}}\texttt{>})} \qquad \text{(MT-Super)}$$

$$\frac{\begin{array}{c}\mathit{mtype}(\texttt{m}, \texttt{N}) = \texttt{<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}_1\wedge\ldots\wedge\overline{\texttt{N}}_n\texttt{>(}\overline{\texttt{S}}_1\wedge\ldots\wedge\overline{\texttt{S}}_n\texttt{)}\rightarrow\texttt{S} \\ \mathit{mtype}(\texttt{m}, \texttt{P}) = \texttt{<}\overline{\texttt{X}} \triangleleft \overline{\texttt{P}}_1\wedge\ldots\wedge\overline{\texttt{P}}_k\texttt{>(}\overline{\texttt{T}}_1\wedge\ldots\wedge\overline{\texttt{T}}_k\texttt{)}\rightarrow\texttt{T}\end{array}}{\mathit{mtype}(\texttt{m}, \texttt{N}\vee\texttt{P}) = \texttt{<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}_1\wedge\ldots\wedge\overline{\texttt{N}}_n\wedge\overline{\texttt{P}}_1\wedge\ldots\wedge\overline{\texttt{P}}_k\texttt{>(}\overline{\texttt{S}}_1\wedge\ldots\wedge\overline{\texttt{S}}_n\wedge\overline{\texttt{T}}_1\wedge\ldots\wedge\overline{\texttt{T}}_k\texttt{)}\rightarrow\texttt{S}\vee\texttt{T}} \qquad \text{(MT-Union)}$$

Method body lookup:

$$\frac{\texttt{class C<}\overline{\texttt{X}}\texttt{<:}\overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{T}}'\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \quad \texttt{<}\overline{\texttt{Y}} \triangleleft \overline{\texttt{P}}\texttt{>}\ \texttt{U m(}\overline{\texttt{U}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{\mathit{mbody}(\texttt{m<}\overline{\texttt{V}}\texttt{>}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = \overline{\texttt{x}}.[\overline{\texttt{T}}/\overline{\texttt{X}}][\overline{\texttt{V}}/\overline{\texttt{Y}}]\texttt{e}_0} \qquad \text{(MB-Class)}$$

$$\frac{\texttt{class C<}\overline{\texttt{X}} \triangleleft \overline{\texttt{N}}\texttt{>} \triangleleft \texttt{D<}\overline{\texttt{T}}'\texttt{>}\ \texttt{\{}\overline{\texttt{S}}\ \overline{\texttt{f}}\texttt{;}\ \overline{\texttt{M}}\texttt{\}} \quad \texttt{m} \notin \overline{\texttt{M}}}{\mathit{mbody}(\texttt{m<}\overline{\texttt{V}}\texttt{>}, \texttt{C<}\overline{\texttt{T}}\texttt{>}) = \mathit{mbody}(\texttt{m<}\overline{\texttt{V}}\texttt{>}, [\overline{\texttt{T}}/\overline{\texttt{X}}]\texttt{D<}\overline{\texttt{T}}'\texttt{>})} \qquad \text{(MB-Super)}$$

Bound of type:

$$\begin{array}{lcl} \mathit{bound}_\Delta(\texttt{X}) & = & \Delta(\texttt{X}) \\ \mathit{bound}_\Delta(\texttt{C<}\overline{\texttt{T}}\texttt{>}) & = & \texttt{C<}\overline{\texttt{T}}\texttt{>} \\ \mathit{bound}_\Delta(\texttt{N}\vee\texttt{P}) & = & \mathit{bound}_\Delta(\texttt{N})\vee\mathit{bound}_\Delta(\texttt{P}) \end{array}$$

Expression typing:

$$\Delta;\Gamma \vdash \texttt{x} : \Gamma(\texttt{x}) \qquad \text{(T-Var)}$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \mathit{ftype}(\mathtt{f}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{T}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{T}} \tag{T-Field}$$

$$\frac{\begin{array}{c}\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \mathit{mtype}(\mathtt{m}, \mathit{bound}_\Delta(\mathtt{T}_0)) = \mathtt{<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}_1 \mathtt{\wedge} \ldots \mathtt{\wedge} \overline{\mathtt{N}}_n\mathtt{>(}\overline{\mathtt{U}}_1 \mathtt{\wedge} \ldots \mathtt{\wedge} \overline{\mathtt{U}}_n\mathtt{)} \mathtt{\to} \mathtt{U}_0 \\ \Delta \vdash \overline{\mathtt{V}} \text{ ok} \quad \Delta \vdash \overline{\mathtt{V}} \mathtt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}_1 \quad \cdots \quad \Delta \vdash \overline{\mathtt{V}} \mathtt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{N}}_n \\ \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{T}} \quad \Delta \vdash \overline{\mathtt{T}} \mathtt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{U}}_1 \quad \cdots \quad \Delta \vdash \overline{\mathtt{T}} \mathtt{<:} [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\overline{\mathtt{U}}_n\end{array}}{\Delta;\Gamma \vdash \mathtt{e}_0\mathtt{.m<}\overline{\mathtt{V}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)} : [\overline{\mathtt{V}}/\overline{\mathtt{X}}]\mathtt{U}_0} \tag{T-Invk}$$

$$\frac{\Delta \vdash \mathtt{C<}\overline{\mathtt{U}}\mathtt{>} \text{ ok} \quad \mathit{fields}(\mathtt{C<}\overline{\mathtt{U}}\mathtt{>}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}} \quad \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \quad \Delta \vdash \overline{\mathtt{S}} \mathtt{<:} \overline{\mathtt{T}}}{\Delta;\Gamma \vdash \mathtt{new\ C<}\overline{\mathtt{U}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)} : \mathtt{C<}\overline{\mathtt{U}}\mathtt{>}} \tag{T-New}$$

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \quad \Delta \vdash \mathtt{T}_0 \mathtt{<:} \mathtt{T}_1 \mathtt{\vee} \mathtt{T}_2 \quad \Delta \vdash \mathtt{T}_1 \mathtt{\vee} \mathtt{T}_2 \text{ ok} \quad \Delta;\Gamma, \mathtt{x}\mathtt{:}\mathtt{T}_1 \vdash \mathtt{e}_1 \mathtt{:} \mathtt{S}_1 \quad \Delta;\Gamma, \mathtt{y}\mathtt{:}\mathtt{T}_2 \vdash \mathtt{e}_2 \mathtt{:} \mathtt{S}_2}{\Delta;\Gamma \vdash \mathtt{case\ e}_0 \mathtt{\ of\ (T}_1 \mathtt{\ x)\ e}_1 \mathtt{\ |\ (T}_2 \mathtt{\ y)\ e}_2 \mathtt{:} \mathtt{S}_1 \mathtt{\vee} \mathtt{S}_2} \tag{T-Case}$$

Method typing:

$$\frac{\begin{array}{c}\Delta = \overline{\mathtt{X}}\mathtt{<:}\overline{\mathtt{N}}\mathtt{,}\overline{\mathtt{Y}}\mathtt{<:}\overline{\mathtt{P}} \quad \Delta \vdash \overline{\mathtt{T}}, \mathtt{T}, \overline{\mathtt{P}} \text{ ok} \quad \Delta;\overline{\mathtt{x}}\mathtt{:}\overline{\mathtt{T}}, \mathtt{this} : \mathtt{C<}\overline{\mathtt{X}}\mathtt{>} \vdash \mathtt{e}_0 \mathtt{:} \mathtt{S} \quad \Delta \vdash \mathtt{S} \mathtt{<:} \mathtt{T} \\ \mathtt{class\ C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>} \mathtt{\lhd D<}\overline{\mathtt{V}}\mathtt{>\ \{\ \ldots\ \}} \quad \mathit{mtype}(\mathtt{m}, \mathtt{D<}\overline{\mathtt{V}}\mathtt{>}) \text{ undefined}\end{array}}{\mathtt{<}\overline{\mathtt{Y}} \mathtt{\lhd} \overline{\mathtt{P}}\mathtt{>\ T\ m(}\overline{\mathtt{T}}\ \overline{\mathtt{x}}\mathtt{)\{\ return\ e}_0\mathtt{;\ \}\ OK\ IN\ C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>}} \tag{T-MethodNew}$$

$$\frac{\begin{array}{c}\Delta = \overline{\mathtt{X}}\mathtt{<:}\overline{\mathtt{N}}\mathtt{,}\overline{\mathtt{Y}}\mathtt{<:}\overline{\mathtt{P}} \quad \Delta \vdash \overline{\mathtt{T}}, \mathtt{T}, \overline{\mathtt{P}} \text{ ok} \quad \Delta;\overline{\mathtt{x}}\mathtt{:}\overline{\mathtt{T}}, \mathtt{this} : \mathtt{C<}\overline{\mathtt{X}}\mathtt{>} \vdash \mathtt{e}_0 \mathtt{:} \mathtt{S} \quad \Delta \vdash \mathtt{S} \mathtt{<:} \mathtt{T} \\ \mathtt{class\ C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>} \mathtt{\lhd D<}\overline{\mathtt{V}}\mathtt{>\ \{\ \ldots\ \}} \\ \mathit{mtype}(\mathtt{m}, \mathtt{D<}\overline{\mathtt{V}}\mathtt{>}) = \mathtt{<}\overline{\mathtt{Y}} \mathtt{\lhd} \overline{\mathtt{P}}\mathtt{>(}\overline{\mathtt{T}}\mathtt{)} \mathtt{\to} \mathtt{T}' \quad \overline{\mathtt{Y}}\mathtt{<:}\overline{\mathtt{P}} \vdash \mathtt{T} \mathtt{<:} \mathtt{T}'\end{array}}{\mathtt{<}\overline{\mathtt{Y}} \mathtt{\lhd} \overline{\mathtt{P}}\mathtt{>\ T\ m(}\overline{\mathtt{T}}\ \overline{\mathtt{x}}\mathtt{)\{\ return\ e}_0\mathtt{;\ \}\ OK\ IN\ C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>}} \tag{T-MethodOvr}$$

Class typing:

$$\frac{\overline{\mathtt{X}}\mathtt{<:}\overline{\mathtt{N}} \vdash \overline{\mathtt{N}}, \mathtt{D<}\overline{\mathtt{S}}\mathtt{>}, \overline{\mathtt{T}} \text{ ok} \quad \overline{\mathtt{M}} \text{ OK IN } \mathtt{C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>}}{\mathtt{class\ C<}\overline{\mathtt{X}} \mathtt{\lhd} \overline{\mathtt{N}}\mathtt{>} \mathtt{\lhd D<}\overline{\mathtt{S}}\mathtt{>\ \{}\overline{\mathtt{T}}\ \overline{\mathtt{f}}\mathtt{;}\ \overline{\mathtt{M}}\mathtt{\}\ OK}} \tag{T-Class}$$

Computation:

$$\frac{\mathit{fields}(\mathtt{C<}\overline{\mathtt{S}}\mathtt{>}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\mathtt{(new\ C<}\overline{\mathtt{S}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{))\mathtt{.}f}_i \longrightarrow \mathtt{e}_i} \tag{R-Field}$$

$$\frac{\mathit{mbody}(\mathtt{m<}\overline{\mathtt{V}}\mathtt{>}, \mathtt{C<}\overline{\mathtt{T}}\mathtt{>}) = \overline{\mathtt{x}}\mathtt{.e}_0}{\mathtt{(new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{))\mathtt{.m<}}\overline{\mathtt{V}}\mathtt{>(}\overline{\mathtt{d}}\mathtt{)} \longrightarrow [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)/this}]\mathtt{e}_0} \tag{R-Invk}$$

$$\frac{\emptyset \vdash \mathtt{C<}\overline{\mathtt{T}}\mathtt{><:} \mathtt{N}_1}{\mathtt{case\ new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)\ of\ (N}_1\ \mathtt{x)e}_1\mathtt{|(N}_2\ \mathtt{y)e}_2 \longrightarrow [\mathtt{new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)/x}]\mathtt{e}_1} \tag{R-LCase}$$

$$\frac{\emptyset \vdash \mathtt{C<}\overline{\mathtt{T}}\mathtt{>} \not<: \mathtt{N}_1 \quad \emptyset \vdash \mathtt{C<}\overline{\mathtt{T}}\mathtt{><:} \mathtt{N}_2}{\mathtt{case\ new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)\ of\ (N}_1\ \mathtt{x)e}_1\mathtt{|(N}_2\ \mathtt{y)e}_2 \longrightarrow [\mathtt{new\ C<}\overline{\mathtt{T}}\mathtt{>(}\overline{\mathtt{e}}\mathtt{)/y}]\mathtt{e}_2} \tag{R-RCase}$$