情報処理学会論文誌　プログラミング　Vol. 2　No. 2　105–131 (Mar. 2009)

型安全な再利用可能アスペクトを目指したMJベースのAOP言語とその型システムの提案

草野 直樹[†1]　鎌田 十三郎[†1]

ソフトウェア開発の効率化においては，関心事の分離が重要である．MJでは，汎用クラス宣言において，その型パラメータのメンバ情報に応じたメンバを宣言できる．この機能はモーフィングと呼ばれ，各クラスに応じた同様のコードを，まとめて記述可能である．また，型パラメータを任意のクラスで具体化しても，型エラーが起きないことを判定する型システムを持つ．一方で，AOP言語で行われているような，コールサイトに分散するコードフラグメントをまとめることや，複数のクラスのメンバをまとめて宣言することを，MJはサポートしていない．本論文は，アスペクトを安全に再利用可能にすることを目指し，MJに基づくAOP言語と，その型システムの提案を行う．型システムの特徴は，アスペクトを任意のプログラムに適用した際に，メソッド追加により，メンバシグネチャの重複やオーバライド違反が起きないことを保証する点である．また，アスペクトに対応するために，ユニーク名の命名機構を新たに導入し，それにともなう型規則の拡張を行った．

Proposal of a MJ-based AOP Language and Its Type System for Type-safeness of Reusable Aspects

Naoki Kusano[†1] and Tomio Kamada[†1]

The separation of concerns is important for efficient software constructions. MJ provides a technique called “morphing” for specifying generic classes whose members are produced by iterating over members of other classes given by type parameters, with avoiding similar codes for individual members. MJ also has a type system that judges whether morphing classes are type safe on arbitrary instantiations of type parameters. On the other hand, MJ does not support AOP-like features to bring together code fragments spread over call-sites or declarations of members for multiple classes. This paper proposes an AOP core language based on MJ and its type system for safely reusable aspects. Our type system ensures that applications of the aspect keep the type safeness of various target programs. The novel factor of the type system is checking whether the aspect may pose problems such as conflicts of member signatures or violations in method overriding. To allow description of type-safe aspects, we adopt a new feature of unique naming into our AOP language, and make enhancements of typing rules.

1. はじめに

ソフトウェア開発の効率化のために，**関心事の分離**の研究がさかんに行われている．関心事の分離とは，プログラム中の関連するコードフラグメントをモジュールとして分離することや，同様のコードの繰返しとなる部分をまとめることであり，そのために，AOP言語等が提案されている．

代表的なAOP言語AspectJ[1)]においては，コールサイトに分散するコードフラグメントをまとめることや，複数のクラスのメンバをまとめて宣言するといった，**横断的な関心事**を，アスペクトという形で分離できる．しかし，AOP言語では一般的に，アスペクトを複数のプログラムで利用することが難しく，アスペクトを別プログラムに適用した際には，型エラーが発生してしまう危険性があった．本研究では，アスペクトの再利用を重視し，任意のプログラムで再利用する際の型安全性の判定を，アスペクト作成時に行うことを可能とした．

AOP言語ではないが，型安性に再利用可能なモジュールを目指した研究として，MJ言語[2),3)]がある．MJ言語のモーフィング機能は，汎用クラス宣言において，その型パラメータのメンバ情報に応じたメンバを宣言できる．この機能を利用することで，各クラスに応じた同様のコードをモジュール化できる．また，その型システムでは，汎用クラスを型パラメータを任意のクラスで具体化しても，型エラーが起きないことを判定し，汎用クラス再利用時の型安全性を保証している．ただし，MJ言語はアスペクトの機構を持たず，横断的関心事の分離はサポートできていない．

本論文は，アスペクトを安全に再利用することを目指し，MJベースのAOP言語と，その型システムの提案を行うものである．提案AOP言語・型システムは，MJ言語とその型システムを拡張する形で構築している．MJ言語にアスペクト機構および，重複のないユニークな名前を付ける命名機構を導入し，それにともなう型システムの拡張を行った．型システムでは，アスペクトを任意のMJプログラムに適用しても，変換後の各MJクラスで

†1 神戸大学大学院工学研究科情報知能学専攻
Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University

型エラーを起こさないことを判定する．提案型システムの主な特徴は，アスペクトのメソッド追加により，メンバシグネチャの重複やオーバライド違反を起こさないことを調べる点である．これにより，アスペクト再利用時の型安全性を，アスペクト作成時に保証することに成功した．

本論文の章構成を述べる．まず，2 章で MJ ベースの AOP 言語の提案を行う．3 章では，提案 AOP 言語を用いたアプリケーション事例を紹介する．続く 4 章では型安全性を判定するアプローチについて述べ，5 章で提案 AOP 言語の定式化と型システムの提案を行う．6 章で関連研究との比較を行い，7 章でまとめる．

2. MJ ベースの AOP 言語の提案

まず，本研究のベースとした MJ 言語を紹介し，その後に提案 AOP 言語の説明を行う．

2.1 MJ の紹介

MJ は，Java をベースに拡張された言語である．以下に，MJ のモーフィング機能を利用したクラス宣言の例を示す．`Logger<X>` は，メソッド呼び出しのロギング機能を付加する汎用クラスである．ロギングの対象とする任意のクラスで，型パラメータ `X` を具体化して利用する．

```
1  class Logger<X> extends X {
2    <R, U*> [meth] for(public R  meth(U) : X.methods )
3      public R meth(U u) {
4        R result = super.n(u);
5        System.out.println( "Returned: " + result);
6        return result;
7  } }
```

`Logger<X>` クラスは，`X` のメソッドの中から，パターン　`R meth(U)` に合致するメソッドに対して繰返しを行う，モーフィング機能を利用して実装されている．`R`，`U` および `meth` はパターンマッチ変数である．`R` は任意のクラスに，`meth` は任意の名前に合致する．また，`U` のように `*` 記号を付記して宣言された場合は，任意個の型列（0 個も含む）に合致する[*1]．この例では，任意の戻値クラス・名前・引数型列の，型変数 `X` のメソッドにパターンマッチ[*2]し，同じシグネチャ・名前のメソッドが，`Logger<X>` に宣言される．また，MJ では，`Logger<X>` クラスのように，型パラメータ `X` のサブクラスとなるクラス宣言が可能である．

この `<R, U*> [meth] for(R meth(U):X.methods)` のような修飾を **reflective ブロック**，reflective ブロックで修飾されたメソッド宣言を **reflective なメソッド宣言**と呼ぶ．対して，reflective ブロックで修飾されていないメソッド宣言は，**static なメソッド宣言**[*3]と呼ぶ．reflective なメソッド宣言の実体は，reflective ブロックのパターンマッチ条件に応じて範囲を持つことになり，この範囲を **Range** と呼ぶ．Range は，reflective ブロックのパターンマッチ条件，メソッド名およびマッチ対象となるクラスの組で表される．

たとえば，先の `Logger<X>` クラスを，`Point`，`Thread` クラスで具体化した場合は，

```
1  class Logger<Point> extends Point {
2    public Point getLocation() { ... }
3    public String toString() { ... }
4    ...
5  }
6  class Logger<Thread> extends Thread {
7    public String getName() { ... }
8    ...
9  }
```

のようになり，型パラメータを具体化するクラスごとに実装がまったく別物となり，型パラメータが具体化されるまで，各 `Logger<X>` がどんなメソッドを持つか決まらない．また，型パラメータを継承するクラスであるため，クラス継承関係もまったく異なる．そのため，MJ では，同じ `Logger` クラスであっても，`Logger<Point>` と `Logger<Thread>` クラスのように，型パラメータを異なるクラスで具体化したクラスを，別クラスとして明確に区別している．

MJ 論文で提案された型システムは，primitive 型や `U*` 等を省いた，FGJ[4] レベルのコア言語を対象として作られている．クラスの型チェックを，型パラメータを具体化しない状態で行い，任意のクラスで具体化しても型エラーが起きないことを判定している．

2.2 提案する AOP 言語

我々は，安全に再利用可能なアスペクトの記述を目指した AOP 言語を提案する．対象言語は MJ 言語である．提供する AOP 言語の特徴は，MJ 風の記法を採用し，型パラメータおよび reflective ブロックを用いて，変更を加えるクラス群および分散するコールサイトの指定を表現する点である．また，その型システムの目的は，アスペクトが特定のプログラムに依存せず，再利用可能であることを判定することである．そのために，変更を加える対象クラス群を具体化しない状態で，メンバ ID の衝突やオーバライド違反が起きないことの判定を行う．以下に，アスペクトの例を示す．

```
1  aspect SampleAspect {
2    <X, R, U*> [meth] for(R  meth(U) : X.methods ) {
3      add R X.copy#meth(U u) {
```

⋆1 型変数 `R` は primitive 型には合致しない．ただし，`U*` が合致する型列の要素の 1 つとして primitive 型を含むことは許される．

⋆2 MJ ではこのほかに，修飾子，メソッドの例外宣言，アノテーションのパターンマッチも可能．

⋆3 本論文での「static なメソッド」は，`static` 修飾子を付加したクラスメソッドとは異なる．

```
4         return this.meth$org(u);
5       }
6       exec R X.meth(U u) {
7         R result = proceed();
8         System.out.println( "Returned: " + result);
9         return result;
10    } } }
```

アスペクトの宣言は，クラス宣言と同様の構造で，**aspect** キーワードに続けて，アスペクト名の宣言を行い，続く {} 内に，プログラムへの変更差分を記述する．アスペクトが対処する横断的関心事は一般的に，クラス構造上の関心事「静的横断」と，プログラム動作・振舞い上の関心事「動的横断」に分かれており，我々の AOP 言語では，それぞれに対応する機能として，「クラスへのメンバ追加」と「アドバイス（命令の書き換え）」の機能を持つ．

3–5 行目はクラスへのメンバ追加の例であり，**add** キーワードに続けてメンバの宣言を行っている．追加するメソッドの ID が，追加クラスの既存メソッドと重複しないように，特殊な `copy#meth` という名前を用いている．このように記述することで，**命名機構**により `#` の前の文字列（以降，prefix と呼ぶ）`copy` と，`#` 以降のメンバ名 `meth` の組に対して，重複のない一意な名前（ユニーク名）が与えられる[*1]．追加対象とするクラスは，メンバ名の直前で指定しており，この例では `X` である．`X` は任意のクラスにマッチするので，この例では，2 行目の reflective ブロックと合わせて，任意のクラスに対してそのクラスの各メソッド `meth` に応じた `copy#meth` メソッドを追加する．

アドバイスの例は，6–10 行目で，**exec** キーワードに続けてメンバを宣言している．`exec` はメソッドボディの再定義を意味し，このほかにも，メソッド呼び出し命令の再定義 `call`，フィールドアクセス命令の再定義 `get`，`set` がある．対象とするクラスおよびメンバの指定は，メンバ追加の場合と同様に行う．この例は，`R X.meth(U)` にパターンマッチするメソッドのボディを，このアドバイスのボディ相当に置き換える．

追加メソッドやアドバイスのボディでは，`this` は対象クラス（この例では `X`）に型付けされ，通常のメソッド宣言時と同様にコード記述が可能である．加えて，新規のものとして，アドバイス適用を行わないオリジナル実行のための，`$org` 式および `proceed()` がある．`$org` 式は，`$org` キーワードを用いて，4 行目の `this.meth$org(u);` のように記述することで，`X.meth(U)` メソッド宣言時そのままの処理を行う[*2]．**proceed()** は，アドバイスのボディ中でのみ利用可能で，アドバイス適用対象のオリジナル実行を意味する（7 行目）．先のアスペクトを次の `C` クラスに適用した場合を例に，`$org` 式と **proceed()** の動作を述べる．

```
class C {    C get() { return this;}  … }
```

たとえば，プログラム中に `new C().get();` というメソッド呼び出し式があった場合には，6–10 行目のアドバイスの適用により，ログ出力が行われる．その際の `proceed()` は，`this.get()` 相当のオリジナル実行を行う．もし，`C.get()` に他のアドバイスも適用されるならば，`proceed()` は，それらのアドバイスを適用した後の処理を行う．一方，プログラム中に `new C().get$org();` という `$org` 式があった場合には，すべてのアドバイスを無視して `C.get()` メソッド宣言時そのままの処理を行うため，ログ出力は行われない．

以降は，アスペクト適用の仕様と，仕様上の制限について例を通して説明する．まずは，追加対象クラスの指定法について述べる．

```
1   aspect TargetClsSample {
2     <X, R> [m] for(R m() : X.methods){
3       add R List<X>.pre1#m(){ ... }}        // AM1
4     <X, Y extends List<X>,R> [m] for(R m() : X.methods){
5       add R Y.pre2#m(){ ... }}              // AM2
6     <X, Z, R> [m] for(R m() : X.methods){
7       add R Map<X,Z>.pre3#m(){ ... }}       // AM3
8     <X, Y extends List<X>,R> [m] for(R m() : X.methods){
9       add R Map<X,Y>.pre4#m(){ ... }}       // AM4
10  }
```

ここでは，サンプルコード中の各メソッド宣言を，`AM1`-`AM4` で識別して説明する．`AM1` では，任意のクラスで具体化された `List` クラス（`List<X>` で総称）に，その型引数クラス（`X`）を対象として reflective にメソッドが追加される．たとえば，`List<String>` に，`String` を対象として reflective にメソッドが追加される．`AM2` では，`List<X>` だけでなく，そのサブタイプとなる任意のクラス（`Y` で総称）も追加対象クラスとなる．`AM3` は，型引数が任意のクラスの組となるすべての `Map` クラス（`Map<X,Z>`）に，第 1 型引数を対象として reflective にメソッドが追加される．`AM4` は，`AM3` に似ているが，`Map<X,Y>` の `X` と，`Y` のバウンドに現れる `X` が，共通となる型引数の組であることが要求される．たとえば，`Map<String,ArrayList<String>>` は `AM4` の対象となるが，`Map<String,ArrayList<Integer>>` は `AM4` の対象とはならない．

1 つのメソッドに，複数のアドバイスが指定された場合の動作について述べる．我々の枠組みでは，先に宣言されたアドバイスがより後で適用（効果が優先）されるように優先順位を設け，順次適用してゆく．

```
1   aspect MultipleAdviceSample {
2     <X> [m] for(Object m() : X.methods){
3       exec Object X.m(){
```

⋆1 MJ での，名前に文字列を付加する記法とは，意味付けが異なる．詳しくは 6 章において議論する．
⋆2 `$org` 式は，クラス宣言のメソッドボディでも利用可能である．

```
4         throw new Exception( "Should not called. "); }}  // EM1
5     <X, R> [m] for(R m() : X.methods){
6       exec R X.m(){ ...; return proceed();  }}          // EM2
7     <X, R, U*> [m] for(R m(U) : X.methods){
8       exec R X.m(U u){ ...; return proceed(); }}        // EM3
9   }
```

上記の例では，X にマッチするクラスの，無引数で戻り値 Object のメソッドに対しては，EM1〜EM3 アドバイスがすべて適用される．各アドバイスのパターンマッチ条件は異なり，EM1，EM2，EM3 の順で，制約が緩くなる．このときの動作を説明する．我々の枠組みでは，後で宣言されたアドバイスから順に適用されるため，まず EM3 適用によるコード追加が行われる．次に，EM2 が適用されるが，このとき，オリジナル実行 proceed() には，EM3 適用後のコードが挿入される．最後に，EM1 の適用により，結果，無引数で戻値 Object のメソッドのボディは例外を投げるものとなる．

次に，メソッド追加の無限ループを防ぐために，我々は，アスペクトで追加（add）されたメンバについては，reflective ブロックのパターンマッチ対象としないという制限を設けている（一方，exec 等のアドバイス適用対象になるメンバは，対象クラスの既存メンバなので，パターンマッチの対象から除外されない）．

```
1   aspect Sample {
2     <X,R,U*> [m] for(R m(U) : X.methods){
3       add R X.pre1#m(U u, int x){ ... }}
4   }
5   // 利用例-------------------------
6   Point p;
7    //Point.move(int, int) に対応して追加
8       p.pre1#move(0, 1, 2);
9    //Point.pre1#move(int, int, int) に対応して追加？
10       p.pre1#pre1#move(0, 1, 2, 3); // エラー!!
```

上記の例は，アスペクトにより追加された X.pre#m 相当を呼び出す利用例である．利用例の 1 つ目，p.pre1#move 呼び出しは問題ない．一方で，2 つ目の例のように，アスペクトにより追加された Point.pre1#move は，パターンマッチの対象としないので，p.pre1#pre1#move 呼び出しは許されない．

3. アプリケーション事例

本論文で提案する AOP 言語を用いた，安全に再利用可能なアスペクトの例を紹介する．

3.1 ライブラリクラスの拡張

アスペクトのメソッド追加機能を利用して，ライブラリクラスに，新たな機能を付加するアスペクトの例として，List 系クラスの拡張事例を示す．List 系クラスは，Java 標準 API に属するクラスである[⋆1]．

```
1   aspect ListSort {
2     <X, F extends Comparable> [f] for(F f : X.fields) {
3       add void List<X>.sortBy#f();
4       add void AbstractList<X>.sortBy#f() {
5         Collections.sort(this, new Comparator<X>() {
6                       public int compare(X e1, X e2) {
7                          return e1.f.compareTo(e2.f);
8                       }});
9     } }
10     // 各 primitive 型の X.fields についても上記同様に記述
11   }
12   // 利用例:  ----------------------------------------
13   List<Point> list;
14     list.sortBy#x(); //x 座標で sort
15     list.sortBy#y(); //y 座標で sort
```

このアスペクト ListSort は，List<X> を具体化するクラスの，各フィールドを基準としてソートを行うメソッドを提供する．プログラマは，これらのメソッドを利用してコーディングすることができる．3，4 行目では，add キーワードを用いてメンバの追加を宣言している．名前の重複回避のため，追加メソッドの名前に命名機構によるユニーク名をつけた sortBy#f メソッドを，List<X> クラスに追加する．sort 処理は，Collections.sort メソッドに任せている．このアスペクトを適用することで，利用例のように，List<Point> クラスに対しては，Point クラスの x，y フィールド（各座標の値）でソートを行う sortBy#x，sortBy#y が追加される．

```
1   aspect InvokeAllElementsOfList {
2     <X, R, U*> [m] for(R m(U) : X.methods) {
3       add List<R> List<X>.invokeAll#m(U u);
4       add List<R> AbstractList<X>.invokeAll#m(U u){
5         List<R> result = new ArrayLsit<R>(this.size());
6         // List の全要素に対してメソッドコール
7         for(X o : this) result.add(o.m(u));
8         return result;
9     } }
10     // 戻値が各 primitive 型の X.methods も上記同様に記述
11   }
12   // 利用例:  ----------------------------------------
13   List<Point> list;
14     list.invokeAll#translate(1, 2);//全 Point を座標移動
15     List<String> str = list.invokeAll#toString();
```

InvokeAllElementsOfList アスペクトは，リストの各要素に対してメソッドを呼び出し，その結果をリスト形式で返す invokeAll#m メソッドを，List 系クラスに追加する．利用例のように，List が保持している Point インスタンスの，すべてに対して座標移動

⋆1 Java のソースは GNU GPL 下での改変が許諾されている．

（translate）を行ったり，全要素の toSting の結果を List として取得したりすることができる．3，4 行目で，sortList の例と同様に，reflective ブロックと add キーワードを利用し，メソッド追加を宣言している．この例では，単純に for 文を用いて順次メソッドを呼び出すだけだが，Thread を作って並列で計算したり，分散環境で計算させたりする実装にする等の応用もある．

MJ においても，ここで紹介したものと同様の機能を List 系クラスを直接書き換える方法で実現する例が紹介されている．しかし，ライブラリの利用者が，ライブラリクラスを直接拡張してしまうと，作成したソフトウェアを異なる実行環境で動作させるために，拡張したライブラリもすべて移さなければならないという問題がある．ライブラリクラスを直接書き換えずに，そのサブクラスとして拡張する方法では，ライブラリクラス（List<X> 等）に対して，追加メソッドを呼ぶことができず，ダウンキャストが必要になる．また，ライブラリ側で作られたインスタンス（List.subList() の戻り値の List 等）は，拡張メソッドを利用できないという問題もある．このような理由から，アスペクトとして変更差分を記述することが望ましいと考える．

3.2　慣例的コーディング手法の略記

プログラムの高速化等の際に，memoise 等，慣例的なコーディング手法が繰り返し現れることがある．これらが出現する回数が多い場合は，逐一手動でコーディングするのは面倒である．また，ソフトウェア開発時には，アルゴリズムのみに集中できるように，このような手法をあまり意識せずに済むようにしたい．これに対し，アスペクトのアドバイス機能およびメソッド追加機能を用いて，「アノテーションで目印を付け，アスペクトで変換」という手法が知られている．我々の枠組みでの実現例を，以下に示す．

```
1   aspect Memoise {
2     //@memoise アノテーションにマッチ
3     <X, U, R> [m] for(@memoise R m(U) : X.methods) {
4      // map フィールド追加
5       add private Map<U,R> X.map#m = new HashMap<U, R>();
6      // 戻り値をキャッシュするコードを追加
7       exec R X.m(U u) {
8         R result = map#m.get(u);
9         if(result == null){
10           result = proceed();//オリジナル実行
11           map#m.put(u, result);
12         }
13         return result;
14    } }
15    // 戻値が各 primitive 型の X.methods も上記同様に記述
16  }
17  // 利用例:  ----------------------------------------
18   class Math{  @memoise int fibonacci(int n){ ... }  }
```

この例は，memoise を実現する例である．memoise とは，「引数に対して戻り値が一意に決まる」関数の結果をキャッシュして，同じ計算が 2 度行われることを回避するテクニックである．この Memoise アスペクトは，5 行目で，add キーワード用いて結果をキャッシュするための HashMap を用意し，7–14 行目で，exec キーワード用いて本来のメソッド処理（10 行目の proceed）の前に HashMap を走査するようにメソッドボディを変更する．利用する際は，例のように memoise を行いたいメソッドに対して，目印として @memoise アノテーションを付加するだけでよい．

```
1   aspect Retry {
2     <X, R, U*, E*> [m] for(@Retry R m(U) throws E : X.methods){
3      // 例外を 3 回まで許容し，リトライするように変更
4       exec R X.m(U u) throws E {
5         int count = 0;   // 例外の回数をカウント
6         while(true){
7           try {
8             return proceed()); //オリジナル実行
9           } catch(E e) {
10            //count が 3 以上になったら，例外を投げる
11            if(count >= 3) throw e;
12          }
13          count++;
14      } }
15    // 戻値が各 primitive 型の X.methods も上記同様に記述
16  }
17  // 利用例:  ----------------------------------------
18   class Facade{  @Retry String userInput(){ ... }  }
```

2 つ目の例は，メソッド実行中に例外が発生した際に，やり直しを行う機能を実現する例である．この Retry アスペクトは，4–14 行目で，メソッド m を，例外を 3 回まで許容してやり直すようにコードを書き換える（exec を利用）．利用する場合には，例のように，やり直し機能を適用したいメソッドに対して，@Retry アノテーションを目印として付加すればよい．また，一部やり直しを行わずにメソッド呼び出しを行いたい部分がある場合には，$org 式を利用して，m$org(u) とすることで対処できる．

4.　安全性の議論

本研究の目標であるアスペクトの型安全性とは，アスペクトを任意の MJ プログラムに適用したとしても，適用後のプログラムが型エラーを起こさないことである．型システムでは，MJ クラスが任意のクラスで具体化されたとしても，アスペクト適用によって，メンバ ID 重複やオーバライド違反，および未定義なメソッド呼び出しといった型エラーが発生し

ないことを判定する．その実現に向けての主な課題は，reflective に宣言されるメソッドの実体が不確定なことに起因する次の 3 つである．

- メソッド ID [*1]の重複回避（4.1 節）
- メソッドのオーバライドセーフの判定（4.2 節）
- 呼び出し対象メソッドが必ず存在すること（メソッド呼び出しの正当性）の判定（4.3 節）

本論文では，MJ に対して「アスペクトの導入」「命名機構の導入」「reflective，static なメソッド宣言の混在を許容」という拡張を行う．それにともない，上記の判定をどのように行うかについて，サンプルケースを通してみてゆく．

4.1 ID の重複回避

ID 重複は，クラス内のメソッド宣言どうしだけでなく，アスペクト内の追加対象クラスが同じになるメソッド宣言どうしでも発生する問題である．本節では，アスペクトの事例を通して，ID 重複回避の議論を行う．まずは，単純なアスペクトの例を示す．

```
1  aspect A {
2    <X, R, U*> [m] for(R m(U) : X.methods)
3      add R X.pre#m(U u) { .... }
4  }
```

このアスペクトの追加メソッド宣言は，reflective に宣言されており，型パラメータ X の各メソッドごとに，X へメソッドが加えられる．アスペクトでは，reflective に追加されるメソッドどうしの ID 重複に加え，X が元々持つ既存メソッドと追加メソッドの間の ID 重複が問題になる．前者は，MJ 時と同様に，X を具体化するクラスが well-typed であると仮定すれば，そのクラスの各メソッドシグネチャはユニークであることから，シグネチャをコピーして reflective に追加されるメソッドどうしに ID 重複はないと判定できる（詳細は，MJ 論文[2] 参照）．後者の，既存のメソッドと追加メソッドの ID 重複は，追加メソッドの名前を，ユニーク名 “pre#m” とし，名前空間を分けることで，ID 重複はないと判定できる．既存要素の名前は，プログラマが自由につけるため，確実に ID 重複を回避するには，追加メソッド宣言に命名機構の利用が必要である．以上のことは，フィールド追加においても，同様のことがいえる．

次に，reflective ブロックが複数ある場合について考察する．以下の例は，各 reflective ブロックの Range が排他になることから，安全だといえる例である．

```
1  aspect A {
2    <X, R> [m1] for(R m1() : X.methods)
3      add R X.pre#m1() { .... }
4    <X, R, U> [m2] for(R m2(U) : X.methods)
5      add R X.pre#m2(U u) { .... }
6  }
```

この例では，各 reflective ブロックは reflective ターゲットは X で同じクラス群となるが，パターンマッチ条件が異なる（m1 は無引数のメソッドにマッチ，m2 は 1 引数のメソッドにマッチ）．そのため，両者がマッチするメソッドの ID は排他となり，追加するメソッドどうしも ID 重複がないと判定できる．MJ においても，この例のように，各 reflective ブロックの Range が排他となる場合には，安全といえていた．

命名機構を利用することで，新たに安全だといえるようになった例を以下に示す．

```
1  aspect A {
2    <X, U> [m1] for(String  m1(U) : X.methods)
3      add String X.pre1#m1(U u) { .... }
4    <X, R> [m2] for(R m2(String) : X.methods)
5      add R X.pre2#m2(String s) { .... }
6  }
```

この例では，m1，m2 のメソッド ID が排他にならないが，追加するメソッドで，ユニーク名の prefix にそれぞれ異なる pre1，pre2 を用いているため，メソッド名の重複は起きないと判定できる．

次の例は，reflective なメソッド宣言と，static なメソッド宣言が混ざる例である．

```
1  aspect A {
2    <X extends Sup> [m] for(String m() : X.methods)
3      add String X.pre#m() { .... }
4    <X extends Sup> add String X.pre#foo() { .... }
5  }
6  class Sup {
7    Object foo( ) { .... }
8  }
```

X に追加される pre#m，pre#foo は，共通の prefix: pre が付加されており，かつシグネチャも同じである．よって，ID 重複回避の条件は，prefix を外した名前の排他関係 foo∉ dom(m)（名前 foo が名前変数 m のドメインに含まれないこと）となる．型パラメータがどんなメソッドを持つかは，具体化されるまでは不確定なため，一般的に reflective な m() と static な foo() の間の重複の危険性は否定できない．ただし，この例のように，X のバウンドである Sup クラスが foo メソッドを持つが，そのシグネチャが reflective のマッチ条件（String m()）に一致しない場合は，foo∉ dom(m) といえる（この例では，Sup.foo の戻り値がマッチ条件に合わない）．このように，X をバウンドしているクラスから，メソッドの重複はないと判定できるケースもある．

⋆1 メソッドの ID は，メソッド名と引数型列の組．フィールドの ID は，フィールド名とする．

4.1.1　reflective 対象，追加対象クラス間の制約

4.1 節の例は，X のメソッドごとに，X へメソッドを加えるという，追加対象クラスに対して reflective 対象クラスが一意に決まる例であった．ここでは，まず追加対象・reflective クラスの指定が適切でないために，型エラーを引き起こす可能性のある事例を示す．

```
1  aspect UnSafeAspect {
2    <X, Y, V, W*> [m] for(V m(W) : Y.methods)
3      add V X.pre#m(W w) { ... }
4  }
```

この例では，reflective の対象である Y と，追加対象である X の間に関連性がなく，任意のクラスの組合せが考えられる．そのため，X に相当するあるクラスに対して，Y 相当クラスが複数存在することになる．結果，複数の Y 相当クラスに対応して，pre#m メソッドを追加することになるため，メソッドの ID 重複の危険がある．問題点は，追加対象クラス X 相当のあるクラスに対して，reflective 対象クラス Y が一意に決まらないことである．このほか，型パラメータの対応がとれず，ID 重複の危険がある例をいくつか以下に示す．

```
1  aspect UnSafe{
2   // X 相当は複数あり，それらのメソッドが単一クラスに追加
3    <X, R> [m] for(R m() : X.methods)
4      add R MyCls<>.pre1#m() { ... }
5   // X が一意に決まらず，List<X>相当が多数存在
6    <X, R> [m] for(R m() : List<X>.methods)
7      add R MyCls<>.pre2#m() { ... }
8   // X の対応は取れるが，Y が一意に決まらない
9    <X, Y extends List<X>, R> [m] for(R m() : Map<X,Y>.methods)
10     add R X.pre3#m() { ... }
11  }
```

次に，追加対象クラスに対して reflective クラスが一意に決まる適切な例を，2.2 節の例以外にいくつか示す．

```
1  aspect Safe{
2   // reflective クラスが，具体的なクラス
3    <X, R> [m] for(R m() : MyCls<>.methods)
4      add R X.pre1#m() { ... }
5   // X,Z 両方が追加クラスに現れる
6    <X, Z, R> [m] for(R m() : Map<X, Z>.methods) {
7      add R Map<X,Z>.pre2#m() { ... }
8   // Y extends List<X>より，Y が決まれば X も一意に決まる
9    <X, Y extends List<X>, R> [m] for(R m() : X.methods)
10     add R Y.pre3#m() { ... }
11   // 一つ上同様，Y が決まれば X も一意に決まる
12    <X, Y extends List<X>, R> [m] for(R m() : Map<X, Y>.methods)
13     add R Y.pre4#m() { ... }
14  }
```

これらの例では，reflective 対象クラスに現れる型パラメータが，追加対象クラスにすべて現れている．下 2 つの例は少し特殊で，reflective 対象クラスに現れる型パラメータ X 自体は，追加対象クラスに現れないが，Y のバウンド（List<X>）において X が現れる．このように，reflective 対象クラスに現れる型パラメータが，対象クラスにも現れていればよい．

4.2　オーバライドセーフ判定

Java におけるオーバライドの規則を簡単に示すと，「親クラスに同じ ID(名前&引数) のメソッドが存在するならば，そのシグネチャは同一」[*1]である．MJ においては reflective ブロックや，型パラメータを継承するクラスが存在することにより，型パラメータが具体化されるまで，メンバ一覧は不確定となり，オーバライドの判定が複雑化する．MJ のオーバライド議論は十分ではないと考え，新たに議論をやり直す．

まず，MJ クラスでのオーバライド議論を行う．親クラスのメンバが不確定となる，単純な MJ のクラスの例を示す．

```
1  class C<X> extends X {
2    void  foo(X x) { ... }   // <- warning
3    <R> [m] for(R m() : X.methods )
4      R m() { ... }
5  }
```

この例では，親クラスが型パラメータ X となるため，C<X> が継承するメンバは不確定となる．そのため，C<X>.foo(X) メソッド宣言でオーバライド違反が起きるかどうかは，X を具体化するクラスに依存してしまう．このような状況は，型システムで拒否する．一方，reflective なメソッド宣言 m は，親クラス X の持つメソッドシグネチャをコピーしているため，必ずオーバライドが成立するといえる．このような，具体化されるまでメソッド一覧が不明なクラスでは，「メソッドが存在しないこと」を示すのは困難なため，「親クラスに同名メソッドが存在し，かつシグネチャが同一」がオーバライドセーフ判定の基本となる．

ただし，同名のメソッドが存在するといえない場合でも，reflective ブロックの Range の関係から，オーバライド違反がないと判定できる場合もある．

```
1  class Sup<X> extends Object {
2    <R> [m] for(R m(String) : X.methods )
3      R m(String s) { ... }
4  }
5  class Sub<Y> extends Sup<Y> {
6    <R> [m1] for(R m1() : Y.methods )
7      List<R> m1() { ... }
8    <U> [m2] for(String m2(U) : Y.methods )
9      String m2(U u) { ... }
10 }
```

*1 Java5.0 以降のオーバライド規則では，戻値型をサブタイプとすることも可能だが，本論文では簡単のため許さないこととする．

この例は，Sup<Y> クラスの Y が具体化されるまで，親クラス Sub<Y> のメンバは不確定である．この場合に，親クラスで宣言されている全メソッドに対して，「Range が排他になる」か「Range の共通部において，メソッドシグネチャが同一」のいずれかが成立することでも，オーバライド違反はないと判定できる．この例ならば，まず，Sup.m，Sub.m1 メソッド宣言の Range は排他となるため，両者の間でオーバライド違反は起きないといえる．次に，Sup.m，Sub.m2 メソッド宣言の Range については，両方にマッチするメソッド（String u(String)）が仮に存在した場合にも，Sup，Sub が持つメソッドは同一シグネチャとなる（この場合，マッチしたメソッドのシグネチャをコピーしているため）．よって，オーバライド違反は起きないと判定できる．

また，親クラスが具体化され，そのメソッド一覧が判明しているときには，「親クラスに同名メソッドが存在しない」ことでオーバライド違反がないと判定できる．

```
1  class C<X> extends X {
2    <R> [m] for(R m() : X.methods )
3      R m() { ... }
4  }
5  class D<Y> extends C<Object> {
6    void foo() { ... }
7  }
```

この例では，C<X> を具体化する Object のメソッド一覧が分かっており，Object.foo() メソッドは存在しないため，C<Object> クラスにも foo() メソッドは存在しないといえる．このため，D<Y>.foo においてオーバライド違反は起きないと判定できる．

まとめると，クラスにおけるオーバライドセーフ判定は以下のどれかが成立すればよい．

- 親クラスに，名前・引数型・戻値型が同一となるメソッドが存在するといえる．
- 親クラスの全メソッド宣言に対し，Range が排他，あるいは Range の共通部で引数型・戻値型が一致．
- 親クラスが具体化された状況で，親クラスに同名メソッドが存在しない．

次に，アスペクトにおけるオーバライドセーフの判定について考察する．アスペクトでは，追加メソッド宣言の名前がユニーク名であることを義務付けることで，クラス宣言とは名前空間を分けているので，クラスで宣言されたメソッドとアスペクトで追加するメソッドの間でオーバライド関係はない．よって，アスペクトで追加するメソッド間のオーバライド関係のみを調べればよい．クラス時の議論との違いは，アスペクトのメソッド追加宣言が，複数のクラスに影響を及ぼす点である．

```
1  aspect SafeAspect {
```

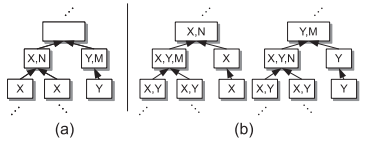

図 1　追加クラス間のクラス階層関係による場合分け
Fig. 1　Possible class hierarchies (1).

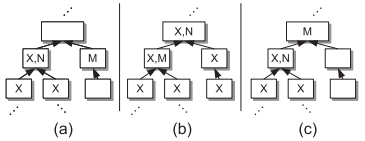

図 2　追加クラス間のクラス階層関係による場合分け 2
Fig. 2　Possible class hierarchies (2).

```
2    <X exnteds N, R> [m] for(R m() : X.methods )
3      add R X.pre#m() { ... }
4    <Y extends M> add String Y.pre#foo() { ... }
5    // <Y extends M> add Y Y.pre#bar() { ... } // エラー
6  }
```

ここでは，N，M は，クラスを表す（型変数ではない）ものとする．この例は，N のサブクラス群 X に pre#m メソッドを，M のサブクラス群 Y に pre#foo メソッドを追加する例であり，N，M の間のクラス階層関係によって，**図 1** に示す 2 つの状況に場合分けできる．図 1 (a) のように N，M に継承関係がない場合は，X，Y が同一クラスになることはないため，ID 重複・オーバライドについて考慮する必要はなく，この例は安全と判定できる．また (b) のように M と N 間にサブタイプ関係がある場合，Y と X には共通部分が存在し，そこでは同一クラスに，両方メソッド追加宣言からメソッドが追加されるため，ID 重複の判定が必要になる．この例では，X をバウンドしているクラス N から，両メソッド追加宣言の ID が排他といえる場合には，オーバライド（違反）は発生しないといえる．

加えて，アスペクトでは，1 個のメソッド追加宣言だけでオーバライド違反が起きる場合もある．上記の例でコメントアウトした pre#bar のように，M のサブクラス群 Y に対して，戻値型も Y となるメソッドを追加する場合，Y 相当の各クラスに追加されるメソッドの戻値型が異なるため，型システムはオーバライド違反であると判定する．

また，以下の例のように，追加対象クラス指定の一方が，型変数でないクラス（M 等）になる場合は，**図 2** のように M，N の継承関係により 3 つの状況がある．

```
1  aspect SafeAspect {
2    <X extends N, R> [m] for(R m() : X.methods )
3      add R X.pre#m() { ... }
4    add String M.pre#foo() { ... }
5  }
```

(a), (b) については，前述と同様である．(c) については，N が M のサブクラスになり，M∉X なので ID 重複は発生しないものの，X は M のサブクラスとなるため，オーバライド違反について考慮する必要がある．この例では，両メソッド宣言において，オーバライドセーフであるためには，クラスのオーバライド議論における Range の判定と同様に，「Range が排他になる」か「Range の共通部において，追加メソッドのシグネチャが同一」のいずれかがいえればよい（この場合，foo∉ dom(m)，あるいは X に String foo() メソッドが存在することを示せればよい）．

アスペクトの，オーバライドセーフについてまとめると以下のようになり，いずれかが成立すればよい．

- 各々の追加対象クラスにサブタイプ関係がない．
- 各々の追加メソッドの ID が排他．
- 各々の追加対象クラスに重複はないが，サブタイプになる可能性があるとき，メソッド宣言の Range からオーバライド違反がないといえる．

4.3　メソッド呼び出しの正当性

繰返しになるが，モーフィングを利用したクラスでは，型パラメータを具体化するまでそのメンバ一覧は不確定である．そのため，プログラム中のメソッド呼び出しに対して，対応するメソッド宣言が必ず存在することを保証しなければならない．

まずは，クラスで宣言されたメソッドを呼び出す場合について，MJ での判定法を示す．

```
1  class Decl<Y> {
2    <V, W*> [m] for(V m(W) : Y.methods)
3      V m(W w) { ... }
4  }
5  class Ref<X> {
6    Decl<X> dx;
7    <U*> [n] for(String n(U) : X.methods)
8      String n(U u) { return dx.n(u); }
9  }
```

この例は，reflective に宣言されているメソッド Decl<Y>.m(W) 相当を，異なる reflective ブロック中から呼び出している（8 行目の dx.n(u)）．この dx.n(u) 相当のメソッド宣言が，Decl<X> クラスのメソッド m 相当の中に含まれており，引数や戻り値の型が一致する保証が必要である．MJ では，reflective ブロックの Range の包含関係と，その際の型変数の対応関係から型安全性の判定を行っている．今回の場合，Ref 側の Range は，Decl 側の Range の一部分（戻り値 V が String に限定された場合）で，dom(n)⊂dom(m) といえる．加えて，Range の共通部分（n=m）においては，X の同じメソッドをパターンマッチするので，V=String，W=U という型変数の対応が成立する．以上のことから，String Decl<X>.n(U) 相当が存在し，dx.n(u) 呼び出し時の引数・戻値型が整合だと，MJ では判定していた．

これを基にして，アスペクトで追加したメソッドを呼び出す場合の型安全性判定を考察する．

```
1  aspect A {
2    <Y, Z extends Marker<Y>, V, W*> [m] for(V m(W) : Y.methods)
3      add V Z.pre#m(W w) { ... }
4  }
5  class Ref<X> extends Marker<X>{
6    <U*> [n] for(String n(U) : X.methods)
7      String n(U u) { return this.pre#n(u); }
8  }                    //class Marker の定義は割愛
```

この例は，アスペクトを用いて Marker のサブクラスにメソッド（pre#m）を追加し，Ref クラスの reflective ブロックの中から，追加したメソッド相当を呼び出している（this.pre#n(u)）．この pre#n(U) メソッド相当が，アスペクトのメソッド追加宣言 Z.pre#m(W) によって Ref<X> クラスに追加されており，かつ引数や戻り値の一致する保証が必要である．クラスで宣言されたメソッドを呼び出す場合との違いは，追加対象クラス（Z）に，呼び出し対象クラス（Ref<X>）が含まれるかの判定も必要になる点である．また，呼び出されるメソッド宣言において prefix が付加されている場合は，共通な prefix（pre）を外した上で，Range の包含関係を調べる．今回の例では，Ref<X> が，追加対象クラス Z（Marker のサブクラス群）の一部である（Ref<X>⊂dom(Z)）．加えて，メソッド Range の包含関係（dom(n)⊂dom(m)）もあり，型変数の対応関係（V=String，W=U）から，型安全であると判定できる．正確には，「呼び出し対象クラス ⊂ 追加対象クラス」という関係は必要なく，呼び出し対象クラスのスーパタイプの 1 つが，追加対象クラスに含まれればよい．

最後に，prefix が複数付加されたメソッド呼び出しを行う状況と，その安全性判定について述べる．

```
1  class Inner<Y> {
2    <V, W*> [m1] for(V m1(W) : Y.methods)
3      V in#m(W w) { ... }
4  }
5  class Wrap<X> {
6    <R, U*> [m2] for(R m2(U) : X.methods)
7      R wrap#m2(U u) { ... }
8  }
9  //利用例----------------------------------------
```

```
10    new Wrap<Inner<String>>().wrap#in#toString();
```

この `Wrap<Inner<String>>.wrap#in#toString()` のように複数 prefix の状況での安全性判定は，これまでのメソッド Range による判定を繰り返す形で行う．まず，`Wrap<Inner<String>>` クラスは，`Wrap<X>` の型変数 `X` を，`Inner<String>` クラスで具体化したクラスであり，`Inner<String>` の各メソッドに，prefix: `wrap` を付加したメソッドを持つ．よって，呼び出すメソッド名から `wrap` を外した `in#toString()` メソッドが `Inner<String>` クラスに存在するといえればよい．`Inner<String>` クラスに `in#toString()` メソッドが存在することは，`in` を外した `toString()` メソッドが `String` クラスに存在することから導けるので，このメソッド呼び出しが正しいと判定できる．

5. 定式化と型システム

MJ 論文[2] では，FGJ [4),5)] に基づいて，MJ 言語のコアとなるサブセット言語 FMJ を定式化し，FMJ を対象とした型システムが提案されていた．本論文においても，型システムの構築に向け，簡単のために，提案した AOP 言語のコアとなるサブセットを抽出し，定式化を行う．基本的に FMJ 基づいており，FMJ からの拡張部分は，ハイライトして示す．定式化に際し，簡単のため `aspect` 宣言は 1 つのみとし，その機能はメソッド追加とメソッド本体再定義（Around execution アドバイス）のみとする．フィールド追加は，メソッドの場合と同様の議論の繰返しになると思われるため，省略する．また，FMJ と同様に，型列変数 $T*$，オーバロード，および共変戻り値（オーバライド時に戻り値をサブタイプ化）も取り扱わない．

5.1 構　　文

文法を，**図 3** に示す．記法は FMJ，FGJ に準じる．C，D は固定のクラス名，X，Y は型変数を表す．N，P，Q は非型変数，S，T，U，V，W は型（非型変数・型変数を区別しない）を表す．f はフィールド名，m はメソッド名，x は引数名を表す．メソッド名 m は命名機構の prefix: pre が 0 個以上付加された名前を表し，prefix を付加されていない名前は n で表す．reflective ブロック Lp における名前変数は u で表す．e は式を表す．なお，`proceed()` は，定式化した言語が関数型であることから，変数の一種 `proceed` として取り扱う．

$\bar{T}$ のような上付き線を付記した場合は，$T_1, T_2, \cdots, T_n$ という要素の列挙を表す．列の連結は : を用いて表し，$\bar{S}{:}\bar{T}$ は，$\bar{S}$ の後に $\bar{T}$ の要素が並ぶことを意味する．列の一要素であることは $\in$ を用いて，$T \in \bar{T}$ のように表す．ルール中で特に意味を持たない値については _ または，その複数形として $\cdots$ で表記する．記号 $\triangleleft$, $\uparrow$ は，それぞれ `extends`，`return` を意味する．メソッドのシグネチャは $\bar{U} \rightarrow U_0$ のように表記し，引数列 $\bar{U}$，戻値 U_0 を意味する．

Syntax :

$T : X \mid N$
$N : C\langle\bar{T}\rangle$
$m : n \mid pre\#m$

$CL : \texttt{class}\ C\langle\bar{X} \triangleleft \bar{N}\rangle \triangleleft T\{\bar{T}\ \bar{f};\ \bar{M}\}$
$M : T\ m(\bar{T}\ \bar{x})\{\uparrow e;\} \mid Lp\ T\ m(\bar{T}\ \bar{x})\{\uparrow e;\}$
$Lp : \langle\bar{X} \triangleleft \bar{N}\rangle\ [\bar{u}]\ \texttt{for}(U\ n(\bar{U}) : T.\texttt{methods})$
$e : x \mid e.f \mid e.m(\bar{e}) \mid \texttt{new}\ C\langle\bar{T}\rangle(\bar{e}) \mid (T)e \mid e.m\texttt{\$org}(\bar{e})$

$A : \texttt{aspect}\ A\{\bar{M}^a\ \bar{M}^e\}$
$M^a : \langle\bar{X} \triangleleft \bar{N}\rangle\ \texttt{add}\ T\ T.m(\bar{T}\bar{x})\{\uparrow e;\} \mid Lp\ \texttt{add}\ T\ T.m(\bar{T}\bar{x})\{\uparrow e;\}$
$M^e : \langle\bar{X} \triangleleft \bar{N}\rangle\ \texttt{exec}\ T\ T.m(\bar{T}\bar{x})\{\uparrow e;\} \mid Lp\ \texttt{exec}\ T\ T.m(\bar{T}\bar{x})\{\uparrow e;\}$

図 3　Syntax
Fig. 3　Syntax.

今回の定式化の目的は，アスペクトを任意のプログラムに適用しても，メソッドの重複・オーバライド違反が発生しないこと，および追加メソッドの参照が正しいことを判定する型システムを構築することである．アスペクト中に現れたクラス（型変数のバウンド等）が，適用対象プログラム中のクラスと不整合を起こさない限り，そのアスペクトの適用によって型エラーが起こらないことを示す．我々の型システムは，FMJ の型システムに基づき，それを拡張する形で構築されている．

プログラムは，*(e,CT,A)* の組で表される．e は main 関数相当の式，CT はクラステーブルでクラス宣言の集合，A はアスペクトである．e，`CT`，`A` すべてが型付けできれば，アスペクトを任意のプログラムに対しても安全に適用可能で，また，アスペクトで追加したメソッドへの参照は正しいといえる．

型付け規則の簡単化のため，FMJ と同様の 2 つの仮定と，新たな 1 つの仮定をおく．FMJ と同様の仮定としては，「パターンマッチ変数が，グローバルにユニーク」と「reflective ブロックで宣言されたパターンマッチ変数のすべてが，パターン宣言または対象クラス指定で利用される」である．新たな仮定は，クラス側とアスペクト側のメソッドの名前重複を避けるために，「クラス宣言とアスペクト宣言では，メソッド宣言に用いるユニーク名の prefix を排他にすること」である．これらの仮定は，構文上で簡単に調べることができる．

5.2 型付け規則

型付け規則の主要部を**図 4**，**図 5** に示す．本節で型システムの概要を述べた後に，次節以

Class typing

$$\frac{\begin{array}{c}\Delta{=}\bar{X}{<:}\bar{N}\quad \Delta{\vdash}\bar{N},T,\bar{T}\ \texttt{ok}\quad \Delta{\vdash}\bar{M}\ \texttt{ok in}\ C{<}\bar{X}{\lhd}\bar{N}{>}\\ \texttt{for all}\ M_i,M_j\in\bar{M}\quad i{\neq}j\\ \texttt{mInfo}(M_i,C{<}\bar{X}{\lhd}\bar{N}{>}){=}(\Delta_i,\_,\_,\_,\langle\Lambda_i,m_i,S_i\rangle)\\ \texttt{mInfo}(M_j,C{<}\bar{X}{\lhd}\bar{N}{>}){=}(\Delta_j,\_,\_,\_,\langle\Lambda_j,m_j,S_j\rangle)\\ \Delta,\Delta_i,\Delta_j{\vdash}\texttt{disjoint}(\langle\Lambda_i,m_i,S_i\rangle,\langle\Lambda_j,m_j,S_j\rangle)\end{array}}{\texttt{class}\ C{<}\bar{X}{\lhd}\ \bar{N}{>}\lhd T\{\bar{T}\ \bar{f};\ \bar{M}\}\ \texttt{ok}}\quad\text{(T-CLS)}$$

Method typing

$$\frac{\begin{array}{c}\texttt{mInfo}(M,C{<}\bar{X}{\lhd}\bar{N}{>}){=}(\Delta',\Gamma,e,\bar{U}{\to}U_0,\langle\Lambda,m,\_\rangle)\quad \Delta',\Delta;\Gamma;\Lambda{\vdash}e\in S\quad \Delta',\Delta{\vdash}S{<:}U_0\\ CT(C){=}\texttt{class}\ C{<}\bar{X}{\lhd}\ \bar{N}{>}\lhd T\{\cdots\}\quad \Delta',\Delta;\Lambda{\vdash}\texttt{override}(m,T,\bar{U}{\to}U_0)\ \texttt{ok in}\ C{<}\bar{X}{>}\end{array}}{\Delta{\vdash}M\ \texttt{ok in}\ C{<}\bar{X}{\lhd}\bar{N}{>}}\quad\text{(T-MTD)}$$

Expression typing

$$\frac{}{\Delta;\Gamma;\Lambda{\vdash}x\in\Gamma(x)}\quad\text{(T-VAR)}$$

$$\frac{\Delta;\Gamma;\Lambda{\vdash}e_0\in T_0\quad \Delta{\vdash}\texttt{fields}(T_0){=}\bar{T}\ \bar{f}}{\Delta;\Gamma;\Lambda{\vdash}e_0.f_i\in T_i}\quad\text{(T-FLD)}$$

$$\frac{\Delta;\Gamma;\Lambda{\vdash}e_0\in T_0\quad \Delta;\Gamma;\Lambda{\vdash}\bar{e}\in\bar{S}\quad \Delta;\Lambda{\vdash}\texttt{mitype}(m,T_0){=}\bar{T}{\to}T\quad \Delta{\vdash}\bar{S}{<:}\bar{T}}{\Delta;\Gamma;\Lambda{\vdash}e_0.m(\bar{e})\in T}\quad\text{(T-INVK)}$$

$$\frac{\Delta{\vdash}C{<}\bar{T}{>}\ \texttt{ok}\quad \Delta;\Gamma;\Lambda{\vdash}\bar{e}\in\bar{S}\quad \Delta{\vdash}\texttt{fields}(C{<}\bar{T}{>}){=}\bar{U}\ \bar{f}\quad \Delta{\vdash}\bar{S}{<:}\bar{U}}{\Delta;\Gamma;\Lambda{\vdash}\texttt{new}\ \ C{<}\bar{T}{>}\ (\bar{e})\in C{<}\bar{T}{>}}\quad\text{(T-NEW)}$$

$$\frac{\Delta;\Gamma;\Lambda{\vdash}e_0\in T_0\quad \Delta{\vdash}T\ \texttt{ok}}{\Delta;\Gamma;\Lambda{\vdash}(T)e_0\in T}\quad\text{(T-CAST)}$$

$$\frac{\Delta;\Gamma;\Lambda{\vdash}e_0\in T_0\quad \Delta;\Gamma;\Lambda{\vdash}\bar{e}\in\bar{S}\quad \Delta;\Lambda{\vdash}\texttt{mitype}(m,T_0){=}\bar{T}{\to}T\quad \Delta{\vdash}\bar{S}{<:}\bar{T}}{\Delta;\Gamma;\Lambda{\vdash}e_0.m\texttt{\$org}(\bar{e})\in T}\quad\text{(T-ORG)}$$

Aspect Typing

$$\frac{\begin{array}{c}\bar{M}^a,\bar{M}^e\ \texttt{ok in}\ A\\ \texttt{for all}\ M_i^a,M_j^a\in\bar{M}^a\quad i{\neq}j\\ \texttt{aInfo}(M_i^a){=}(\Delta_i,\_,\_,\_,T_i,\langle\Lambda_i,m_i,S_i\rangle)\\ \texttt{aInfo}(M_j^a){=}(\Delta_j,\_,\_,\_,T_j,\langle\Lambda_j,m_j,S_j\rangle)\\ \Delta_i,\Delta_j{\vdash}\texttt{disconf}(T_i,\langle\Lambda_i,m_i,S_i\rangle,T_j,\langle\Lambda_j,m_j,S_j\rangle)\end{array}}{\texttt{aspect}\ A\{\bar{M}^a\bar{M}^e\}}\quad\text{(T-ASPECT)}$$

Exec Advice Typing

$$\frac{\texttt{aInfo}(M^e){=}(\Delta,\Gamma,e,\bar{U}{\to}U_0,T,\langle\Lambda,m,S\rangle)\quad \Delta;\Gamma,\texttt{proceed}{\mapsto}U_0;\Lambda{\vdash}e\in S\quad \Delta{\vdash}S{<:}U_0}{M^e\ \texttt{ok in}\ A}\quad\text{(T-EXEC)}$$

Method Addtion Typing

$$\frac{\texttt{aInfo}(M^a){=}(\Delta,\Gamma,e,\bar{U}{\to}U_0,T,\langle\Lambda,\_,\_\rangle)\quad \Delta;\Gamma;\Lambda{\vdash}e\in S\quad \Delta{\vdash}S{<:}U_0\quad T\in dom(\Delta)\ \texttt{implies}\ \Delta{\setminus}(T{<:}N){\vdash}U_0,\bar{U}\ \texttt{ok}}{M^a\ \texttt{ok in}\ A}\quad\text{(T-ADD)}$$

図 4　Typing rules
Fig. 4　Typing rules.

Method Info

$$\frac{M{=}S_0\ m(\bar{S}\ \bar{x}){\{\uparrow e;\}}\quad \Gamma{=}\texttt{this}{\mapsto}C{<}\bar{X}{>},\bar{x}{\mapsto}\bar{S}\quad \bar{X}{<:}\bar{N}{\vdash}S_0,\bar{S}\ \texttt{ok}}{\texttt{mInfo}(M,C{<}\bar{X}{\lhd}\bar{N}{>}){=}(\phi,\Gamma,e,\bar{S}{\to}S_0,\langle\phi,m,C{<}\bar{X}{>}\rangle)}\quad\text{(MInfo-S)}$$

$$\frac{\begin{array}{c}M{=}{<}\bar{Y}{\lhd}\bar{P}{>}\ [\bar{u}]\ \texttt{for}(U_0\ n(\bar{U}){:}X.\texttt{methods})S_0\ m(\bar{S}\ \bar{x}){\{\uparrow e;\}}\\ \Delta{=}\bar{Y}{<:}\bar{P}\quad \Lambda{=}X{\mapsto}\langle\bar{Y},\bar{u},U_0\ n(\bar{U})\rangle\quad m{=}n\mid pre\texttt{\#}n\\ \Gamma{=}\texttt{this}{\mapsto}C{<}\bar{X}{>},\bar{x}{\mapsto}\bar{S}\quad \bar{X}{<:}\bar{N},\Delta{\vdash}X,\bar{P},U_0,\bar{U},S_0,\bar{S}\ \texttt{ok}\end{array}}{\texttt{mInfo}(M,C{<}\bar{X}{\lhd}\bar{N}{>}){=}(\Delta,\Gamma,e,\bar{S}{\to}S_0,\langle\Lambda,m,X\rangle)}\quad\text{(MInfo-R)}$$

Aspect Info

$$\frac{\begin{array}{c}M^*{=}{<}\bar{Y}{\lhd}\ \bar{P}{>}\ \texttt{(add|exec)}\ \ S_0\ T.m(\bar{S}\ \bar{x}){\{\uparrow e;\}}\\ \Gamma{=}\texttt{this}{\mapsto}T,\bar{x}{\mapsto}\bar{S}\quad \Delta{=}\bar{Y}{<:}\bar{P}\quad \Delta{\vdash}\bar{P},T,S_0,\bar{S}\ \texttt{ok}\\ \Lambda{=}\phi{\mapsto}\langle\bar{Y},\phi,\phi\rangle\quad *{=}a\ \texttt{implies}\ m{=}pre\texttt{\#}m'\end{array}}{\texttt{aInfo}(M^*){=}(\Delta,\Gamma,e,\bar{S}{\to}S_0,T,\langle\Lambda,m,T\rangle)}\quad\text{(AInfo-S)}$$

$$\frac{\begin{array}{c}M^*{=}{<}\bar{Y}{\lhd}\ \bar{P}{>}\ [\bar{u}]\ \texttt{for}(U_0\ n(\bar{U}){:}S.\texttt{methods})\texttt{(add|exec)}\ \ S_0\ T.m(\bar{S}\ \bar{x}){\{\uparrow e;\}}\\ \Gamma{=}\texttt{this}{\mapsto}T,\bar{x}{\mapsto}\bar{S}\quad \Delta{=}\bar{Y}{<:}\bar{P}\quad \Delta{\vdash}\texttt{AspOK}(T,S)\\ \Lambda{=}S{\mapsto}\langle\bar{Y},\bar{u},U_0\ n(\bar{U})\rangle\quad \Delta{\vdash}T,S,\bar{P},U_0,\bar{U},S_0,\bar{S}\ \texttt{ok}\\ *{=}a\ \texttt{implies}\ m{=}pre\texttt{\#}n\qquad *{=}e\ \texttt{implies}\ m{=}n\end{array}}{\texttt{aInfo}(M^*){=}(\Delta,\Gamma,e,\bar{S}{\to}S_0,T,\langle\Lambda,m,S\rangle)}\quad\text{(AInfo-R)}$$

Reflective Target Check

$$\frac{}{\Delta{\vdash}\texttt{AspOK}(X,X)}\qquad \frac{\Delta{\vdash}\texttt{AspOK}(X,T_i)}{\Delta{\vdash}\texttt{AspOK}(X,C{<}\bar{T}{>})}$$

$$\frac{\Delta{\vdash}\texttt{AspOK}(X,\Delta(Y))}{\Delta{\vdash}\texttt{AspOK}(X,Y)}\qquad \frac{\forall T_i{\in}\bar{T}\quad \Delta{\vdash}\texttt{AspOK}(T_i,S)}{\Delta{\vdash}\texttt{AspOK}(C{<}\bar{T}{>},S)}$$

Well-formed types

$$\Delta{\vdash}\texttt{Object ok}\quad\text{(WF-OBJECT)}\qquad \frac{X\in dom(\Delta)}{\Delta{\vdash}X\ \texttt{ok}}\quad\text{(WF-VAR)}$$

$$\frac{CT(C){=}\texttt{class}\ C{<}\bar{X}{\lhd}\bar{N}{>}\lhd T\{\cdots\}\quad \Delta{\vdash}\bar{T}\texttt{ok}\quad \Delta{\vdash}\bar{T}{<:}[\bar{T}/\bar{X}]\bar{N}}{\Delta{\vdash}C{<}\bar{T}{>}\ \texttt{ok}}\quad\text{(WF-CLS)}$$

Class subtyping

$$\Delta{\vdash}T{<:}T\quad\text{(S-REFL)}\qquad \Delta{\vdash}X{<:}\Delta(X)\quad\text{(S-VAR)}$$

$$\frac{\Delta{\vdash}T{<:}S\quad \Delta{\vdash}S{<:}U}{\Delta{\vdash}T{<:}U}\quad\text{(S-TRANS)}\qquad \frac{CT(C){=}\texttt{class}\ C{<}\bar{X}{\lhd}\bar{N}{>}\lhd T\{\cdots\}}{\Delta{\vdash}C{<}\bar{T}{>}{<:}[\bar{T}/\bar{X}]T}\quad\text{(S-CLS)}$$

図 5　Sub rules for typing rules
Fig. 5　Sub rules for typing rules.

降で disjoint（**図 6**），override（**図 7**），subsumption（**図 8**）の説明を行う．なお，付属的な関数定義は**図 9** に示す．

本型システムには，FMJ 同様，4 つのマッピング環境が存在する．マッピングは記号↦を用いて表記し，たとえば，$\Delta{=}\cdots,X{\mapsto}N$ ならば，$\Delta(X){=}N$ を意味する．

- Δ：型変数から，そのバウンド型へのマッピング．クラス宣言および reflective ブロックの宣言から構築される（$<\bar{X}\triangleleft\bar{N}>$ から $\bar{X}\mapsto\bar{N}$）．また，型変数はつねに非型変数にバウンドされる．関数 $\mathtt{bound}_\Delta(T)$ は，非型変数 N に対しては $\mathtt{bound}_\Delta(N)=N$，型変数 X に対しては $\mathtt{bound}_\Delta(X)=\Delta(X)$ と定義する．
- Γ：メソッド引数名から，その型へのマッピング．メソッド宣言の引数部から構築される．
- Λ：reflective 対象クラスから，reflective ブロック情報へのマッピング．$<\bar{Y}\triangleleft\bar{P}>[\bar{u}]\ \mathtt{for}(U_0\ n(\bar{U}):T.\mathtt{methods})$ という reflective ブロックより，T から $\langle\bar{Y},\bar{u},U_0\ n(\bar{U})\rangle$ へのマッピングが構築される．$\bar{Y}$，$\bar{u}$ はパターンマッチ型変数，名前変数であり，その後に続くメソッドシグネチャは，パターンマッチの条件である．
- $\mathcal{M}$：パターンマッチ型変数から，マッチする型へのマッピング．2 つのメソッド Range の共通部分（名前が同じ領域）における，各型変数と同値になる型のマッピングを示す．

関数 $\mathtt{mapT}_\mathcal{M}(T)$ は，型 T を，$\mathcal{M}$ でマップした型へ変換する操作を表す（定義は図 9）．

図 4 中の T-CLS，T-MTD（クラス，メソッドの型付け規則）の内容は，基本的に FMJ と同様である．ただし，FMJ と異なり，同一クラス中に static なメソッド宣言と reflective なメソッド宣言が混在することを許しているので，両者を同一の規則で扱えるように関数 `mInfo`（図 5 上部）を定義する．`mInfo` の戻り値は，左から順に，型変数のバウンド環境 Δ，引数型環境 Γ，メソッドボディ e，メソッドシグネチャ $\bar{S}\to S_0$，メソッド宣言の Range $\langle\Lambda,m,S\rangle$ である．reflective なメソッド宣言の Range は，reflective ブロック情報 Λ とメソッド名および reflective 対象クラスの組 $\langle\Lambda,m,X\rangle$ で表す（MInfo-R）．一方 static なメソッド宣言の Range は，メソッド名と当該メソッドが宣言されているクラスの組 $\langle\phi,m,C<\bar{T}>\rangle$ で表す（MInfo-S）．`mInfo` で得られる Range の間での排他性判定を行う `disjoint` 関数の内容も拡張している（説明は 5.3 節）．また，override ルールの改良（説明は 5.4 節）にともない，ルール中で自クラス $C<\bar{X}>$ を参照できるように変更している．

式の型付けについては，T-INVK にて `mitype` 関数（クラスのメソッドだけでなく，アスペクトで追加されるメソッドも Lookup の対象とする）を利用する点と，新たに `org` 式が加わる点が異なる．`org` 式は，アドバイスを適用しないメソッド呼び出しを指示する式で，T-INVK と同様の型付け規則となっている．`mitype` 等のルックアップを行う関数は，呼び出す対象メソッドが存在すると保証できる場合のみ，そのシグネチャを返す．その定義は図 8 に示し，説明は 5.5 節で行う．

今回新たに加わったアスペクトの型付けルールは T-Aspect である．その内容は，メソッド追加（T-Add）とアドバイス（T-Exec）の型付けと，メソッド追加宣言 M^a 間での，メンバシグネチャの衝突・オーバライド違反が起きないこと（disconf）の判定を行っている．static メソッド宣言と reflective なメソッド宣言を共通規格で扱うために，`aInfo`（図 5 中ほど）を定義している．`aInfo` の戻り値は，`mInfo` より 1 つ多く，左から順に，型変数のバウンド環境 Δ，引数型環境 Γ，メソッドボディ e，メソッドシグネチャ $\bar{S}\to S_0$，追加対象クラス T，メソッド宣言の Range $\langle\Lambda,m,S\rangle$ である．また，`AspOK` 関数は，4.1.1 項で議論した，reflective 対象クラスと追加対象クラス間の制約判定を行う．

Method range disjointness

$$\frac{\Lambda_1\vdash\mathtt{static}(m_1)\quad\Lambda_2\vdash\mathtt{static}(m_2)\quad m_1\neq m_2}{\Delta\vdash\mathtt{disjoint}(\langle\Lambda_1,m_1,T_1\rangle,\langle\Lambda_2,m_2,T_2\rangle)}\ \text{(DS-STATIC)}$$

$$\frac{m_1=\overline{pre1\#n_1}\quad m_2=\overline{pre2\#n_2}\quad pre1_i\neq pre2_i}{\Delta\vdash\mathtt{disjoint}(\langle\Lambda_1,m_1,T_1\rangle,\langle\Lambda_2,m_2,T_2\rangle)}\ \text{(DS-UNIQ)}$$

$$\frac{\begin{array}{c}T_1<:T_2\ \ or\ \ T_2<:T_1\\ \left\{\begin{array}{l}\{\Lambda_1\not\vdash\mathtt{static}(m_1)\ \ \mathtt{dropPre}(\langle\Lambda_1,m_1\rangle,\langle\Lambda_2,m_2\rangle)=m_1':m_2'\}or\\ \{\Lambda_2\not\vdash\mathtt{static}(m_2)\ \ \mathtt{dropPre}(\langle\Lambda_2,m_2\rangle,\langle\Lambda_1,m_1\rangle)=m_2':m_1'\}\end{array}\right\}\\ \Delta;\Lambda_1\vdash\mathtt{mtype}(m_1',T_1)=\bar{U}\to U_0\quad\Delta;\Lambda_2\vdash\mathtt{mtype}(m_2',T_2)=\bar{V}\to V_0\\ \bar{Y}=\mathtt{pmVars}(\Lambda_1),\mathtt{pmVars}(\Lambda_2)\quad\Delta;\bar{Y}\not\vdash\mathtt{tunify}(U_0{:}\bar{U},V_0{:}\bar{V})\end{array}}{\Delta\vdash\mathtt{disjoint}(\langle\Lambda_1,m_1,T_1\rangle,\langle\Lambda_2,m_2,T_2\rangle)}\ \text{(DS-TYPE)}$$

$$\frac{\begin{array}{c}\{r{=}1\ and\ s{=}2\}\ \ xor\ \ \{r{=}2\ and\ s{=}1\}\\ \Lambda_r\vdash\mathtt{static}(m_s)\ \ \mathtt{dropPre}(\langle\Lambda_r,m_r\rangle,\langle\Lambda_s,m_s\rangle)=m_r':m_s'\\ \Delta;\Lambda_r\vdash\mathtt{mtype}(m_r',T_r)=\bar{U}\to U_0\quad\Delta;\Lambda_s\vdash\mathtt{mtype}(m_s',T_r)=\bar{V}\to V_0\\ \bar{Y}=\mathtt{pmVars}(\Lambda_1),\mathtt{pmVars}(\Lambda_2)\quad\Delta;\bar{Y}\not\vdash\mathtt{tunify}(U_0{:}\bar{U},V_0{:}\bar{V})\end{array}}{\Delta\vdash\mathtt{disjoint}(\langle\Lambda_1,m_1,T_1\rangle,\langle\Lambda_2,m_2,T_2\rangle)}\ \text{(DS-RANGE)}$$

図 6　Method range disjointness
Fig. 6　Method range disjointness.

5.3　名前のユニークネス

型システムにおける，主なチャレンジの 1 つ目は，図 6 に示す名前のユニークネスの判定である．4.1 節で議論した内容をルール化し，命名機構や Range を用いた，名前の排他性（`disjoint`）判定を行う．今回は FMJ と異なり，reflective なメソッド宣言と static なメソッド宣言が，1 つのクラスに混在することを許すため，reflective な名前と static な名前の間の `disjoint` 判定も行う．disjoint は，クラスのタイピング規則（T-CLS）および，後述するオーバライド判定（disconf）において利用される．`disjoint` の引数は，メソッド宣言の Range 情報 $\langle\Lambda,m,S\rangle$ が 2 つであり，両 Range の名前が，排他になることを判定する．判定規則は 4 種あり，そのいずれかが成立すればよい．1 つ目の DS-STATIC は最も簡単

Method override

$$\frac{\Delta;\Lambda\vdash \texttt{mtype}(m,T)=\bar{V}\to V_0 \quad \bar{V}=\bar{U} \quad V_0=U_0}{\Delta;\Lambda\vdash \texttt{override}(m,T,\bar{U}\to U_0)\ \texttt{ok in}\ T_0} \tag{OVR-SUCCESS}$$

$$\frac{\Lambda\vdash \texttt{static}(m)\ \ \texttt{inst}(C\!<\!\bar{T}\!>)\ \ \Delta;\Lambda\not\vdash \texttt{mtype}(m,C\!<\!\bar{T}\!>)}{\Delta;\Lambda\vdash \texttt{override}(m,C\!<\!\bar{T}\!>,\bar{U}\to U_0)\ \texttt{ok in}\ T_0} \tag{OVR-INSTANT}$$

$$\frac{\begin{array}{l}\Lambda=S\mapsto\langle \_,\_,\_\rangle \ \textit{or}\ \{\Lambda=\phi\ \textit{and}\ S=T_0\}\\ CT(C)=\texttt{class}\ C\!<\!\bar{X}\lhd\ \bar{N}\!>\lhd T\{\cdots\bar{M}\}\\ \Delta;\Lambda\vdash \texttt{override}(m,[\bar{T}/\bar{X}]T,\bar{U}\to U_0)\ \texttt{ok in}\ T_0\\ \texttt{for all}\ M_i\in\bar{M}\\ \quad \texttt{mInfo}(M_i,C\!<\!\bar{X}\lhd\bar{N}\!>)=(\Delta',\_,\_,\_,\langle\Lambda',m',S'\rangle)\\ \quad \Delta_k=\Delta,[\bar{T}/\bar{X}](\Delta')\quad \Lambda_k=[\bar{T}/\bar{X}](\Lambda')\quad S_k=[\bar{T}/\bar{X}]S'\\ \quad \Delta_k\vdash \texttt{disconf}(T_0,\langle\Lambda,m,S\rangle,\ C\!<\!\bar{T}\!>,\langle\Lambda_k,m',S_k\rangle)\end{array}}{\Delta;\Lambda\vdash \texttt{override}(m,C\!<\!\bar{T}\!>,\bar{U}\to U_0)\ \texttt{ok in}\ T_0} \tag{OVR-RANGE}$$

Class instantiated

$$\frac{CT(C)=\texttt{class}\ C\!<\!\bar{X}\lhd\ \bar{N}\!>\lhd T\{\cdots\bar{M}\}\quad \texttt{inst}([\bar{T}/\bar{X}]T)\qquad \forall M_i\in\bar{M}\ \ \texttt{mInfo}(M_i,C\!<\!\bar{X}\lhd\bar{N}\!>)=(\_,\_,\_,\_,\langle\Lambda,m,S\rangle),\ \ S\in\bar{X}\ \ \texttt{implies inst}([\bar{T}/\bar{X}]S)}{\texttt{inst}(C\!<\!\bar{T}\!>)}$$

Disconflict

$$\frac{\bar{Z}=\texttt{Var}_\Delta(\Lambda_1,T_1),\texttt{Var}_\Delta(\Lambda_2,T_2)\quad \text{no}\ \mathcal{M}\quad \forall Z_i\in\bar{Z},\quad Z_i\mapsto T\in\mathcal{M}\quad \Delta;\bar{Z}\vdash T\prec:Z_i\quad \Delta\vdash \texttt{mapT}_\mathcal{M}(T_1)<:\texttt{mapT}_\mathcal{M}(T_2)\ \textit{or}\ \Delta\vdash \texttt{mapT}_\mathcal{M}(T_2)<:\texttt{mapT}_\mathcal{M}(T_1)}{\Delta\vdash \texttt{disconf}(T_1,\langle\Lambda_1,m_1,S_1\rangle,T_2,\langle\Lambda_2,m_2,S_2\rangle)} \tag{DSC-CLS}$$

$$\frac{\Delta\vdash \texttt{disjoint}(\langle\Lambda_1,m_1,S_1\rangle,\langle\Lambda_2,m_2,S_2\rangle)}{\Delta\vdash \texttt{disconf}(T_1,\langle\Lambda_1,m_1,S_1\rangle,T_2,\langle\Lambda_2,m_2,S_2\rangle)} \tag{DSC-NAME}$$

$$\frac{\bar{Z}=\texttt{Var}_\Delta(\Lambda_1,T_1),\texttt{Var}_\Delta(\Lambda_2,T_2)\quad \Delta;\bar{Z}\not\vdash \texttt{tunify}(T_1,T_2)\quad \left\{\Delta;\mathcal{M}\vdash \texttt{munify}(\langle\Lambda_1,m_1,S_1\rangle,\langle\Lambda_2,m_2,S_2\rangle)\ \textit{or}\ \mathcal{M}=\phi\right\}\quad \Delta;\Lambda_1\vdash \texttt{mitype}(m_1,T_1)=\bar{U}\to U_0\quad \Delta;\Lambda_2\vdash \texttt{mitype}(m_2,T_2)=\bar{V}\to V_0\quad \mathcal{M}\vdash U_0{:}\bar{U}=V_0{:}\bar{V}}{\Delta\vdash \texttt{disconf}(T_1,\langle\Lambda_1,m_1,S_1\rangle,T_2,\langle\Lambda_2,m_2,S_2\rangle)} \tag{DSC-RANGE}$$

Method unify

$$\frac{\left\{\ \texttt{dropPre}(\langle\Lambda_1,m_1\rangle,\langle\Lambda_2,m_2\rangle)=m_1':m_2'\ \textit{or}\ \texttt{dropPre}(\langle\Lambda_2,m_2\rangle,\langle\Lambda_1,m_1\rangle)=m_2':m_1'\ \right\}\quad \Delta;\Lambda_1\vdash \texttt{mtype}(m_1',S_1)=\bar{U}\to U_0\quad \Delta;\Lambda_2\vdash \texttt{mtype}(m_2',S_2)=\bar{V}\to V_0\quad S_1<:S_2\ \ \textit{or}\ \ S_2<:S_1\quad \bar{Y}=\texttt{pmVars}(\Lambda_1),\texttt{pmVars}(\Lambda_2)\quad \Delta;\mathcal{M};\bar{Y}\vdash \texttt{tunify}(U_0{:}\bar{U},V_0{:}\bar{V})}{\Delta;\mathcal{M}\vdash \texttt{munify}(\langle\Lambda_1,m_1,S_1\rangle,\langle\Lambda_2,m_2,S_2\rangle)}$$

図 7　Rules for method override
Fig. 7　Rules for method override.

Field lookup

$$\Delta\vdash \texttt{fields}(\texttt{Object})=\cdot \tag{FT-OBJ}$$

$$\frac{CT(C)=\texttt{class}\ C\!<\!\bar{X}\lhd\ \bar{N}\!>\lhd T\{\bar{S}\ \bar{f};\cdots\}\quad \Delta\vdash \texttt{fields}(\texttt{bound}_\Delta([\bar{T}/\bar{X}]T))=\bar{U}\ \bar{g}}{\Delta\vdash \texttt{fields}(C\!<\!\bar{T}\!>)=\bar{U}\ \bar{g},[\bar{T}/\bar{X}]\bar{S}\ \bar{f}} \tag{FT-CLS}$$

Method lookup

$$\frac{\Lambda(T)=\langle\bar{Y},\bar{u},U_0\ n(\bar{U})\rangle}{\Delta;\Lambda\vdash \texttt{mtype}(n,T)=\bar{U}\to U_0} \tag{MT-Λ}$$

$$\frac{\Delta\vdash T<:T'\quad T\neq T'\quad \Delta;\Lambda\vdash \texttt{mtype}(m,T')=\bar{U}\to U_0}{\Delta;\Lambda\vdash \texttt{mtype}(m,T)=\bar{U}\to U_0} \tag{MT-SUP}$$

$$\frac{\begin{array}{c}CT(C)=\texttt{class}\ C\!<\!\bar{X}\lhd\ \bar{N}\!>\lhd T\{\cdots\bar{M}\}\quad M_i\in\bar{M}\\ \texttt{mInfo}(M_i,C\!<\!\bar{X}\lhd\bar{N}\!>)=(\Delta',\_,\_,\bar{S}\to S_0,\langle\Lambda',m',S'\rangle)\\ \Delta_k=\Delta,[\bar{T}/\bar{X}](\Delta')\quad \Lambda_k=[\bar{T}/\bar{X}](\Lambda')\quad S_k=[\bar{T}/\bar{X}]S'\\ \Delta_k;\mathcal{M}\vdash \texttt{subsump}(C\!<\!\bar{T}\!>,\langle\Lambda_k,m',S_k\rangle,C\!<\!\bar{T}\!>,\langle\Lambda,m,S_k\rangle)\end{array}}{\Delta;\Lambda\vdash \texttt{mtype}(m,C\!<\!\bar{T}\!>)=[\bar{T}/\bar{X}](\texttt{mapT}_\mathcal{M}(\bar{S})\to\texttt{mapT}_\mathcal{M}(S_0))} \tag{MT-CLS}$$

InterType lookup

$$\frac{\begin{array}{c}\texttt{aspect}\ A\{\cdots\bar{M}^a\}\quad M_i^a\in\bar{M}^a\\ \texttt{aInfo}(M_i^a)=(\Delta',\_,\_,\bar{S}\to S_0,T',\langle\Lambda',m',S'\rangle)\\ \Delta',\Delta;\mathcal{M}\vdash \texttt{subsump}(T',\langle\Lambda',m',S'\rangle,T,\langle\Lambda,m,S'\rangle)\end{array}}{\Delta;\Lambda\vdash \texttt{itype}(m,T)=\texttt{mapT}_\mathcal{M}(\bar{S})\to\texttt{mapT}_\mathcal{M}(S_0)} \tag{IT}$$

$$\frac{\Delta\vdash T<:T'\quad T\neq T'\quad \Delta;\Lambda\vdash \texttt{itype}(m,T')=\bar{U}\to U_0}{\Delta;\Lambda\vdash \texttt{itype}(m,T)=\bar{U}\to U_0} \tag{IT-SUP}$$

Method and InterType lookup

$$\frac{\Delta;\Lambda\vdash \texttt{mtype}(m,T)=\bar{U}\to U_0\ \ \textit{or}\ \ \Delta;\Lambda\vdash \texttt{itype}(m,T)=\bar{U}\to U_0}{\Delta;\Lambda\vdash \texttt{mitype}(m,T)=\bar{U}\to U_0} \tag{MTIT}$$

Subsumption

$$\frac{\Lambda_1\vdash \texttt{static}(m)\quad \bar{Z}=\texttt{Var}_\Delta(\Lambda_1,T_1)\quad \Delta;\mathcal{M};\bar{Z}\vdash \texttt{tunify}(T_1,T_2)}{\Delta;\mathcal{M}\vdash \texttt{subsump}(T_1,\langle\Lambda_1,m,S\rangle,T_2,\langle\Lambda_2,m,S\rangle)} \tag{sump-S}$$

$$\frac{\begin{array}{c}\Lambda_1\not\vdash \texttt{static}(m_1)\quad \bar{Z}=\texttt{Var}_\Delta(\Lambda_1,T_1)\quad \Delta;\mathcal{M}';\bar{Z}\vdash \texttt{tunify}(T_1,T_2)\\ S'=\texttt{mapT}_{\mathcal{M}'}(S)\quad \texttt{dropPre}(\langle\Lambda_1,m_1\rangle,\langle\Lambda_2,m_2\rangle)=m_1':m_2'\\ \Delta;\Lambda_1\vdash \texttt{mtype}(m_1',S)=\bar{U}\to U_0\quad \Delta;\Lambda_2\vdash \texttt{mtype}(m_2',S')=\bar{V}\to V_0\\ \bar{Y}=\texttt{pmVars}(\Lambda_1)\quad \Delta;\mathcal{M};\bar{Y}\vdash \texttt{tunify}(T_1{:}U_0{:}\bar{U},T_2{:}V_0{:}\bar{V})\end{array}}{\Delta;\mathcal{M}\vdash \texttt{subsump}(T_1,\langle\Lambda_1,m_1,S\rangle,T_2,\langle\Lambda_2,m_2,S\rangle)} \tag{sump-R}$$

図 8　Lookup functions
Fig. 8　Lookup functions.

で，static な名前どうしでその名前が異なる場合である．2 つ目の DS-UNIQ は，各名前の prefix が不一致となる場合である．メソッド名 m を，prefix の列 $\overline{pre}$ と n に分け，prefix の列の各要素を比較している．3 つ目の DS-TYPE は，reflective 対象クラス間に継承関係があり，かつそのパターンマッチ条件が排他になるといえる場合である．最後の 4 つ目の DS-RANGE は，static な名前と reflective な名前の間の排他性判定で，reflective 側のクラス T_r に，static な名前のメソッド m_s が存在し，そのシグネチャが，パターンマッチ条

Drop prefix

$$\frac{\Lambda_1{=}\phi \quad \langle\Lambda_1, m_1\rangle \sqsupseteq_{id} \langle\Lambda_2, m_2\rangle}{\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda_2, m_2\rangle){=}m_1 : m_2}$$

$$\frac{\Lambda_1{=}\_ \mapsto \langle \_, \_, U_0\ n_1(\bar{U})\rangle \quad \langle\Lambda_1, n_1\rangle \sqsupseteq_{id} \langle\Lambda_2, m_2\rangle}{\texttt{dropPre}(\langle\Lambda_1, n_1\rangle, \langle\Lambda_2, m_2\rangle){=}n_1 : m_2}$$

$$\frac{\Lambda_1{=}\_ \mapsto \langle \_, \_, U_0\ n_1(\bar{U})\rangle \quad \langle\Lambda_1, n_1\rangle \sqsupseteq_{id} \langle\Lambda_2, n_2\rangle}{\texttt{dropPre}(\langle\Lambda_1, pre\#n_1\rangle, \langle\Lambda_2, pre\#m_2\rangle){=}n_1 : m_2}$$

Method Name Subrange

$$\frac{\Lambda_1 \vdash \texttt{static}(n_1)\ \texttt{implies}\ \ \Lambda_2 \vdash \texttt{static}(m_2)\ \texttt{and}\ n_1{=}m_2}{\langle\Lambda_1, n_1\rangle \sqsupseteq_{id} \langle\Lambda_2, m_2\rangle} \qquad \frac{\langle\Lambda_1, m_1\rangle \sqsupseteq_{id} \langle\Lambda_2, m_2\rangle}{\langle\Lambda_1, pre\#m_1\rangle \sqsupseteq_{id} \langle\Lambda_2, pre\#m_2\rangle}$$

Static name

$$\frac{m{=}\overline{pre\#n} \quad \Lambda{=}T \mapsto \langle \_, \bar{u}, \_\rangle\ \texttt{implies}\ n \notin \bar{u}}{\Lambda \vdash \texttt{static}(m)}$$

Pattern Matching Variables

$$\texttt{pmVars}(\phi){=}\phi \qquad \texttt{pmVars}(\_ \mapsto \langle \bar{Y}, \_, \_\rangle){=}\bar{Y}$$

$$\frac{\bar{X}{=}\texttt{pmVars}(\Lambda) \quad \Delta(Z){=}N \quad \texttt{Var}_\Delta(\Lambda, N){=}\bar{Y}}{\texttt{Var}_\Delta(\Lambda, Z){=}(Z \cup \bar{Y}) \cap \bar{X}} \qquad \frac{\bar{X}{=}\texttt{pmVars}(\Lambda) \quad \bar{Z}{=}\bigcup_{T_i \in \bar{T}} \texttt{Var}_\Delta(\Lambda, T_i)}{\texttt{Var}_\Delta(\Lambda, C{<}\bar{T}{>}){=}\bar{Z} \cap \bar{X}}$$

Type Equality

$$\mathcal{M} \vdash T{=}T \qquad \frac{T \mapsto S \in \mathcal{M}\ \ or\ \ S \mapsto T \in \mathcal{M}}{\mathcal{M} \vdash T{=}S} \qquad \frac{\mathcal{M} \vdash \bar{T}{=}\bar{S}}{\mathcal{M} \vdash C{<}\bar{T}{>}{=}C{<}\bar{S}{>}}$$

Type mapping

$$\frac{X \notin dom(\mathcal{M})}{\texttt{mapT}_\mathcal{M}(X){=}X}\ \text{(TM-Var1)} \qquad \frac{\mathcal{M}(X){=}T}{\texttt{mapT}_\mathcal{M}(X){=}\texttt{mapT}_\mathcal{M}(T)}\ \text{(TM-Var2)} \qquad \frac{\texttt{mapT}_\mathcal{M}(\bar{T}){=}\bar{S}}{\texttt{mapT}_\mathcal{M}(C{<}\bar{T}{>}){=}C{<}\bar{S}{>}}\ \text{(TM-CLS)}$$

図 9　Auxiliary definitions
Fig. 9　Auxiliary definitions.

件に合わない場合である．DS-TYPE/RANGE におけるパターンマッチ条件の不一致判定は，FMJ 同様に「`tunify` が成立しない」ことで判定する（説明は 5.6 節）．reflective メソッド宣言時に加えられたユニーク名の prefix を，外したうえで行う必要があり，これに対応するのは `dropPre` 関数である（定義は，図 9）．第 1 引数の $\langle\Lambda_1, m_1\rangle$ の組を見て，m_1 に prefix が加えられていない場合はそのままの名前を返し，prefix が加えられている場合は m_2 からも同じ prefix を外した名前を返す．同時に，`dropPre` 中の$\sqsupseteq_{id}$によって，メソッド名の包含関係の可能性も調べている．

アスペクトで追加するメソッドとすでにクラスで宣言されているメソッドとの名前重複は，命名機構の利用を義務付けることで対処している．`aInfo` 関数定義において，メソッド追加宣言（$*{=}a$ の場合）に対しては，その追加メソッド名が必ずユニーク名であることを要求する．アスペクトで追加するメソッドどうしの名前重複については，オーバライド違反のチェックと合わせて `disconf` が判定する（T-ASPECT）．

5.3.1　`AspOK` 関数

`AspOK` 関数は，4.1.1 項で議論した，reflective 対象と追加対象のクラス間の制約判定を行う．第 1 引数は reflective 対象クラス，第 2 引数は追加対象クラスである．reflective 対象クラスが型パラメータ X の場合，追加対象クラスが直接 X になるか，追加対象クラスの型引数かバウンドの中に X が現れることを要求する．reflective 対象クラスが $C{<}\bar{T}{>}$ 形の場合は，型引数に現れる型パラメータすべてについて，`AspOK` の成立を要求する．

5.4　オーバライドセーフ

型システムにおける，次のチャレンジは，図 7 に示すオーバライドセーフの判定である．型パラメータを任意のクラスで具体化した際でも，オーバライド違反が起きないことを調べる．4.2 節の議論を規則化し，クラスの場合の判定は `override` 規則，アスペクトの場合の判定は `disconf` 規則が対応する．`override` の引数は，左から順に，メソッド名，親クラス，メソッドシグネチャである．環境 Λ は，判定対象となるメソッド宣言の reflective 情報で，T_0 はメソッドが宣言されたクラスである．クラスのオーバライドセーフ判定は 3 つの規則から行われる．1 つ目の OVR-SUCCESS ルールでは，親クラスにおいて，同名メソッドが見つかり，そのシグネチャが一致することでオーバライドが必ず成立することを判定している．なお，親クラスのメソッドと，さらにその親クラスとの間でのオーバライド違反については，親クラスの全メソッドに対しても `override` 判定を行うため問題ない．2 つ目の OVR-INSTANT ルールは，親クラスのメンバ一覧が確定しており（`inst`$(C{<}\bar{T}{>})$ 成立），その中に m が存在しないことを判定している．`inst`（定義は図 7 の中ほど）は，引数にとるクラスが具体化されメンバ一覧が確定していることを判定する．その内容は，自クラスのメンバ一覧に影響を与える「親クラス」および「reflective 対象クラス」のすべてについて，メンバ一覧が確定している（`inst` が成立する）ことである．3 つ目の OVR-RANGE ルールでは，メソッド宣言の Range から，「名前が同じになる場合にはそのシグネチャも一致する」という性質を `disconf` の DSC-NAME/RANGE を用いて判定している．この性質が親クラスの全メソッド宣言との間で成立し，かつさらに親クラスを遡りながら `override` が成立すれば，オーバライド違反がないと判定する．

`disconf` 関数は，OVR-RANGE だけでなく，アスペクトのオーバライドセーフ判定にも用いる．その引数は，追加対象クラス T，メソッド宣言 Range $\langle\Lambda, m, S\rangle$ が 2 つずつで

ある．disconf には 3 つのルールがある．1 つ目の DSC-CLS は，どんな $\mathcal{M}$ においても，追加対象クラス間 T_1，T_2 に継承関係が存在しないことを判定している．なお $\mathtt{Var}_\Delta(\Lambda, T)$ は，Λ で宣言された型変数の中から，クラス T で使われている物だけを取り出す操作を表す（定義は，図 9）．$T \prec: Y$ という表記は，型変数 Y が T をマッチできることを示す（定義は MJ 論文参照）．2 つ目の DSC-NAME は，メソッド名が排他になることを disjoint を用いて判定している．3 つ目の DSC-RANGE は，追加対象クラス T_1，T_2 に重複がないこと（tunify の不成立）と，追加メソッドシグネチャが一致することを判定する．$\mathcal{M}$ は munify で構築されたメソッドシグネチャの束縛環境か，あるいは空とする．$\mathcal{M}$ が空の状況でも，たとえばメソッドシグネチャが固定（引数なし，戻り値 String 等）ならば，$m_1{=}m_2$ の場合にオーバライドが成立するためである．なお disconf は，override から利用される場合，T_1，T_2 は，メソッドが宣言されたクラスとその親クラスという，確実に継承関係にあるクラスの組となるため，DSC-CLS は成立せず，DSC-NAME/RANGE のみの判定となる．

5.5　メソッドコールの正しさ

型システムの最後のチャレンジ，3 つ目は，メソッド呼び出しに対応するメソッド宣言が，必ず存在することの保証である．4.3 節の議論では，呼び出す対象メソッドが，クラスで宣言されたメソッドの場合は「メソッド Range の包含関係の判定」，アスペクトで追加されるメソッドの場合は「クラスの包含関係の判定+メソッド Range の包含関係の判定」であった．対応する規則は，図 8 の下部の Subsumption であり，クラス・アスペクトどちらにも利用できるようにしている．subsump の引数は，対象クラス T，メソッド宣言情報 $\langle \Lambda, m, S \rangle$ が 2 つずつである．クラスメソッドを対象として利用する場合は，subsump 呼び出し時の引数の T_1，T_2 を同値とすることで，クラスの包含関係判定 $\mathtt{tunify}(T_1, T_2)$ が必ず成立（$\mathcal{M}{=}\phi$）し，メソッド宣言 Range の包含関係のみが判定できる．

subsump 規則は，図 8 中ほどのメソッドのルックアップ関数で利用される．mtype はクラスで宣言されたメソッドを lookup し，itype はアスペクトで宣言された追加メソッドを lookup する．これらは，呼び出される対象メソッドが存在すると保証できる場合のみ，そのシグネチャを返す．なお，reflective に宣言されたメソッドが呼び出される対象となる場合は，型のマッピング環境 $\mathcal{M}$ を用いて，呼び出し側の型に変換したシグネチャを返す．メソッド呼び出し式の型付け（T-INVK）に，これらの関数を用いることで，メソッド呼び出しの正しさを判定する．なお，T-INVK 規則で実際に用いられている mitype ルールは，mtype と itype の，いずれかルックアップできた方の結果を返す．このとき，mtype，itype が同時に成立しないことは，クラス定義部とアスペクト定義部で名前空間を排他にしていることからいえる．

Type unification

$$\frac{\mathcal{M} \vdash \bar{T}{=}\bar{S} \quad \forall Y_i \in \bar{Y}, \quad Y_i \mapsto T \in \mathcal{M} \quad \Delta; \bar{Y} \vdash T \prec: Y_i}{\Delta; \mathcal{M}; \bar{Y} \vdash \mathtt{tunify}(\bar{T}, \bar{S})}$$

図 10　Type unification
Fig. 10　Type unification.

5.6　パターンマッチ

図 10 に示す tunify は，型列の単一化を行う関数であり，単一化が可能な場合は，マッチ環境 $\mathcal{M}$ を構築する．$\bar{Y}$ は，pmVars，Var 関数を用いて取得される．Var にて現れる $\cup$ 記号は，列を集合的なものとして考えて集合の和を作る操作を意味し，$\cap$ 記号は集合の共通部分をとる操作を意味する．

tunify は，パターンマッチ型変数 $\bar{Y}$ に対する型の割当て環境 $\mathcal{M}$ において，引数列 $\bar{T}$，$\bar{S}$ を等しくできるか判定する．どんな $\mathcal{M}$ に対しても，引数列を等しくできない場合を，$\Delta; \bar{Y} \not\vdash \mathtt{tunify}(\bar{T}, \bar{S})$ と表記する．

tunify には 2 種の利用法がある．1 つ目は，subsumption における包含関係の判定である．この場合は，包含する側のパターンマッチ型変数のみを受け取り，マッチ環境 $\mathcal{M}$ が構築できることで，包含関係があると判定する．もう一方は，disjoint，disconf 等における，Range の排他性判定である．こちらの場合は，両方のパターンマッチ型変数を受け取り，マッチ環境 $\mathcal{M}$ が作れない（Range の共通部分が存在しない）ときに，両者の Range が排他であると判定する．

なお，tunify の定義は，MJ から特に拡張はないため，詳細は MJ 論文を参照のこと．

5.7　健　全　性

我々の提案した AOP 言語および型システムにおいて，MJ 時と同様の Theorem，Lemma が成立する．付録で，提案 AOP 言語の Reduction 規則の紹介（A.1）および，健全性の証明（A.2）を行う．Reduction 操作は記号 $\rightarrow$ を用いて表す．

Theorem 1（Subject Reduction）
$\Delta; \Gamma; \Lambda \vdash e \in T$ かつ $e \rightarrow e'$ ならば，$\Delta; \Gamma; \Lambda \vdash e' \in S$ かつ $\Delta \vdash S <: T$ なる S が存在．

Theorem 2（Progress）
$\phi; \phi; \phi \vdash e \in T$ なる e が，well-typed なとき，
1) 式 e がその中に $\mathtt{new}\ C{<}\bar{T}{>}\ (\bar{e}).f$ を持つとき，$\mathtt{fields}(C{<}\bar{T}{>}){=}\bar{U}\ \bar{f} \quad f{=}f_i$．

2) 式 e がその中に $\texttt{new}\ C\texttt{<}\bar{T}\texttt{>}(\bar{e}).m(\bar{d})$ を持つとき，$\phi;\phi \vdash \texttt{mibody}(m, C\texttt{<}\bar{T}\texttt{>}) = (\bar{x}, e_0) \quad \texttt{\#}(\bar{x}) = \texttt{\#}(\bar{d})$ [⋆1].

3) 式 e がその中に $\texttt{new}\ C\texttt{<}\bar{T}\texttt{>}(\bar{e}).m\texttt{\$org}(\bar{d})$ を持つとき，$\phi;\phi \vdash \texttt{mibody}(m, C\texttt{<}\bar{T}\texttt{>}) = (\bar{x}, e_0) \quad \texttt{\#}(\bar{x}) = \texttt{\#}(\bar{d})$.

4) 式 e がその中に $\texttt{new}\ C\texttt{<}\bar{T}\texttt{>}(\bar{e}).m(\bar{d})$ を持つとき，

$\phi;\phi \vdash \texttt{LookupA}(m, C\texttt{<}\bar{T}\texttt{>}) = \bar{\alpha} \quad \forall \alpha_i \in \bar{\alpha}, \quad \alpha_i = (\bar{x}, e_0) \quad \texttt{\#}(\bar{x}) = \texttt{\#}(\bar{d})$.

Theorem 3（Type Soundness）

$\phi;\phi;\phi\ \vdash e \in T$ なる e を，$e \rightarrow^* e'$ によりノーマルフォームへ簡約した e' は，変数 v $(\phi;\phi;\phi \vdash v \in S, \phi \vdash S <: T)$，または $(T)\texttt{new}\ C\texttt{<}\bar{T}\texttt{>}(\bar{e})$ を含んだ式 $(\phi \vdash T <: C\texttt{<}\bar{T}\texttt{>})$ のいずれかになる．

Lemma 1（Name Uniqueness）

1.1： $\texttt{class}\ C\texttt{<}\bar{X} \triangleleft \bar{N}\texttt{>} \triangleleft T\{\cdots \bar{M}\}\ \texttt{ok}$ において，$\Delta;\Lambda \vdash \texttt{mtype}(m, C\texttt{<}\bar{T}\texttt{>}) = \bar{U} \rightarrow U_0$ を成立させる $M_i (\in \bar{M})$ は 2 つ以上は存在しない．

1.2： $\texttt{aspect}\ A\{\cdots \bar{M}^a\}\ \texttt{ok}$ において，$\Delta;\Lambda \vdash \texttt{itype}(m, C < \bar{T} >) = \bar{U} \rightarrow U_0$ を成立させる $M_i^a (\in \bar{M}^a)$ は 2 つ以上は存在しない．

1.3： $\Delta;\Lambda \vdash \texttt{mitype}(m, C < \bar{T} >) = \bar{U} \rightarrow U_0$ において，$\Delta;\Lambda \vdash \texttt{itype}(m, C < \bar{T} >)$ と $\Delta;\Lambda \vdash \texttt{mtype}(m, C\texttt{<}\bar{T}\texttt{>})$ は同時には成立しない．

Lemma 9　$\Delta \vdash T\ \texttt{ok}$，$\Delta;\Lambda \vdash \texttt{mitype}(m, T) = \bar{U} \rightarrow U_0$ ならば，$\Delta \vdash S <: T$ かつ $\Delta \vdash S\ \texttt{ok}$ なる S について，$\Delta;\Lambda \vdash \texttt{mitype}(m, \texttt{bound}_\Delta(S)) = \bar{U} \rightarrow U_0$ が成立．

4 章で示した型安全性判定の課題は，次の Theorem，Lemma から保証できる．

- ID の重複回避（Lemma 1）：
  クラスおよびアスペクトが well-typed な場合には，$\texttt{mitype}(m, C\texttt{<}\bar{T}\texttt{>})$ を成立させるメソッド宣言が，一意に決定できることから，クラス中に同じ ID（名前）のメソッドは複数存在しない（ID 重複がない）といえる．
- オーバライドセーフの判定（Lemma 9）：
  メソッドルックアップ $\texttt{mitype}(m, T)$ が成立した場合には，T のサブクラス S からも，同じシグネチャでルックアップが成立する．
- メソッド呼び出しの正当性（Theorem 2）：
  Theorem 2 の 2)，3) が，メソッド呼び出しの正当性を保証する．式 e が well-typed な場合には，その中に含まれるメソッド呼び出し式，$\texttt{\$org}$ 式の呼び出し対象となるメソッド宣言が存在（mitype 成立）する．

⋆1 $\texttt{\#}(\bar{x})$ は，要素列 $\bar{x}$ の個数を意味する．

5.7.1　証明方針について

本論文の証明方針は，Reduction 時にアスペクトの適用を行うこととし，型付けできていれば，Reduction が正常に動作すること（Theorem 1）を証明するものである．

別方針として，アスペクトの内容を織り込んだ等価な MJ プログラムへと変換する規則を定義し，変換後の MJ プログラムが well-typed となることを示す方法も考えられる．その場合，変換規則の実現に際し，今回のアスペクトの仕様上，add の実現が課題となる．以下の例を用いて説明する．

```
1  class C<X> { ... }
2  aspect A {
3    add String C<N>.pre#foo() { ... }
4    add Integer C<M>.pre#foo() { ... }
5  }
```

アスペクトの add 操作では，クラスの型パラメータがどのクラスで具体化されているかを含めて，メソッド追加の対象になるかどうかを判定している．そのため，上記の例では，同じ C クラスであっても，C<N>，C<M> に，まったく異なる pre#foo メソッドを宣言しなければならない．しかし，MJ 言語では，型パラメータの具体化に応じた処理を，単一汎用クラス C<X> 上に素直に表現することはできない．このため，現時点では，変換後プログラムを，well-typed な汎用 MJ クラスとして表現可能かどうかを含め，よく分かっていない．

あるいは，FJ のプログラムへ変換する方法も考えられる．この場合は，具体化された汎用クラスそれぞれにクラス宣言を生成（C<N> と C<M> それぞれを，別々のクラスとして宣言）することで，上記の例のような add 操作も実現可能であろう．また，exec アドバイスは，適用対象メソッドのボディを上書きし，$org 式については，オリジナルメソッドを別名でコピーして呼び出せば，実現可能であろう．このような変換規則を定義したうえで，「アスペクトが well-typed ならば，変換規則によって生成される各 FJ クラスが well-typed」という性質を証明することになる．

6. 関 連 研 究

6.1　MJ との差分

本論文の貢献をまとめると，大きく分けて「アスペクトの導入」「命名機構の導入」「reflective，static メンバの混在」の 3 つである．

アスペクト機能とモーフィング機能は，1 つのコードがシグネチャの異なる複数のメソッドに影響するという点で，同じことを指向した機能である．相違点は，モーフィングの効果

範囲が 1 個のクラスに閉じているのに対し，アスペクトでは複数のクラスにわたって効果があるという点である．そのため，アスペクトの型安全性判定に向けて，適用するクラス間の継承関係についての考慮が必要となった．加えて，既存メンバ/追加メンバ間の ID 重複を否定するための命名機構や，reflective/static なメソッド宣言の間での disjoint や disconflict の判定も必要となった（4 章参照）．提案した型システムでは，アスペクトとモーフィングに共通して利用できる規則を用意することで，型システムの肥大化を低減させている．

MJ 論文では，*pre#m* という記法により，名前 *m* の前に，単純に文字列 *pre* を付け加える機構が提案されていた．この機構は，利用しても名前重複の危険があるため，MJ の型システムではサポート対象としていなかった．一方，我々の命名機構は，同じ記法であるが，prefix ごとに名前空間を分離して重複がない名前をつける機構と定義し，簡単かつ確実に名前重複を回避する手段を提供した．

reflective，static なメンバの混在は，disjoint が成立する状況がほとんどないことから，MJ の型システムはサポートしていなかった．本研究では，命名機構の導入および，型変数をバウンドしているクラスが持つメソッドを考慮することで disjoint が成立する DS-RANGE 規則を新たに加えたことで，disjoint を成立させる状況が増えている．これにより，実用的に reflective，static メンバを混在させることを可能としている．これにともない，オーバライドの判定規則を，一から作り直している．

MJ は現在，実際に利用できる言語 MorphJ [3] が配布されている．MorphJ には，同じ著者が提案した cJ 言語[6] の機構をとり入れる等，モーフィングの記述力を向上させる方向での機能拡張が施され，アンチパターン（パターンに一致しないことを，パターンマッチの条件とする）や，reflective ブロックをネストして利用することが可能な言語となっている．

6.2　他の AOP 言語との比較

既存の AOP 言語の中には，3 章で示したようなアスペクトの記述が可能なものはあるが，型安全に再利用できることを保証するシステムはなかった．

代表的な Java 向けの AOP 言語である AspectJ [1] では，アスペクトによるメンバ追加はサポートされているが，本研究における static なメンバの追加のみが可能で，reflective なメンバ追加はできない．また，AspectJ 向けの型システムの研究 [7)–12)] が行われているが，アドバイス適用（動的横断）のみを対象としている．これらは，アドバイス適用により型の不整合が発生しないことの判定および，適合するジョインポイントが存在しえない無意味なポイントカットの検出を行う．我々の提案した型システムのようにメンバ追加（静的横断）もサポートしている型システムの研究はあまり行われていない．

AOP 言語 Sally [13] は，クラスの構造に応じたメンバ追加を可能とし，reflective なメンバ追加相当も実現できる．しかし，命名機構に相当する機能がない等，再利用に向けた対策はなく，型安全性の責任はプログラマが負う．その代わりに，より柔軟なプログラムの書き換えを可能としており，親クラス・実装インタフェースの変更や，Prolog 風の記法を用いた詳細なポイントカット指定等が可能である．

6.3　MetaProgramming

Scheme の Hygienic Macro と本研究の命名機構は，名前重複を回避して再利用性を高めるという目的が同じである．Hygienic Macro では，Macro 内でのローカルな変数の名前を，Macro の展開場所ごとに，重複しない名前に自動で書き換える．命名機構との違いとしては，付けられた名前を外部から参照するかどうかという点があげられる．Hygienic Macro では，Macro 内に閉じたローカル変数の名前を Macro 外から参照することはないが，本研究では，ユーザコードから命名機構によるユニーク名を付けたメンバへの参照が可能である．また，命名機構では，オーバライドをするために同じ名前を割り振ることを要求できる点も異なる．

論文 14) では，Traits の安全性判定を行う型システムが提案されている．Traits では，AOP 言語と同様の，メンバ追加およびメソッドボディの再定義といった，クラス拡張が可能である．ただし，命名機構に相当する機構はなく，名前重複やオーバイライド違反について，Traits 作成時には判定できない．追加メンバの名前を，Traits の利用者が適切に設定し，利用時に型安全性判定を行う型システムとなっている．また，AOP 言語と異なり，複数のコードサイトをまとめて変換する機構はない．

6.4　再利用可能なアスペクトの研究

Transaction や並列性制御等の用途に向けた再利用可能アスペクトを提案する研究[15),16)] が行われている．ただし，我々が提案した型システムのように，アスペクトを安全に再利用可能か判定するシステムの提案は行われていない．

論文 15) は，AspectJ の abstract ポイントカット機能を利用した，再利用可能アスペクトを提案している．提案したアスペクトの安全性については，著者らにより保証されている．

論文 16) では，再利用可能なアスペクトを記述するためのフレームワークへの要求機能についての議論も行われている．その中で，アスペクトが複数ある場合の課題が示唆されている．他のアスペクトで提供された機能を利用するアスペクトや，他のアスペクトが変換する（あるいは変換しない）命令を対象としたアドバイス等，依存関係があるアスペクトどうしに，明示的に依存関係を表す機構が必要になるとの指摘をしている．我々の型システムでは，簡単のためアスペクトを 1 つに限定（アスペクトが複数ある状況を，すべてを集約した

1 個のアスペクトとして抽象化）した定式化の下で議論しているため，アスペクトが複数ある状況を想定できていない．今後は，再利用可能なアスペクトが複数ある状況で，複合的に利用可能かどうかについても考えていきたい．また，その他の要求機能「インスタンスごとのアスペクト適用」，「Run-Time に行うアスペクトの On/Off」および「Thread ごとのアスペクト適用」については，現在対応していない．

7. ま と め

本論文では，アスペクトを安全に再利用可能にすることを目指し，MJ ベースの AOP 言語と，そのコア言語に対する型システムを提案した．提案 AOP 言語では，MJ 言語のモーフィングと同様に，reflective ブロックを用いてアスペクトを記述する．その型システムでは，記述したアスペクトを任意のプログラムに適用しても，メンバ ID の重複やオーバライド違反が起きないことを，アスペクト作成時点で判定することを可能としている．

また，提案した型システムをアスペクトをともなわない MJ 言語の型システムとしてみた場合でも，ユニーク名の命名機構の導入により，reflective，static メンバの混在を可能としている点で向上している．

提案 AOP 言語を用いた再利用可能なアスペクトのサンプルとして，Java 標準 API クラスである `List` クラスを拡張するアスペクトや，memoise テクニックを容易に利用可能とするアスペクトの事例を紹介した．今後は，アスペクトの記述力向上に向けて，より多くのアプリケーション事例の検討を行い，言語仕様を深めていきたい．

謝辞　まず，有益な助言をいただきました査読者の方々に深く感謝いたします．また，当研究室にて，本研究の基本構想となる研究[17] を行った川上祐介氏に感謝します．最後に，本研究は平成 18 年科学研究費補助金若手研究（B）18700029 の補助を受けたものである．

参 考 文 献

1) Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G.: An Overview of AspectJ, *ECOOP 2001 — Object-Oriented Programming*, Lecture Notes in Computer Science, Vol.2072, pp.327–355 (2001).
2) Huang, S.S., Zook, D. and Smaragdakis, Y.: Morphing: Safely Shaping a Class in the Image of Others, *ECOOP 2007 — Object-Oriented Programming*, Lecture Notes in Computer Science, Vol.4609, pp.399–424 (2007).
3) Huang, S.S. and Smaragdakis, Y.: Expressive and safe static reflection with MorphJ, *PLDI '08: Proc. 2008 ACM SIGPLAN conference on Programming language design and implementation*, pp.79–89, ACM (2008).
4) Igarashi, A., Pierce, B. and Wadler, P.: Featherwieght Java: A minimal core calculus for Java and GJ, *OOPSLA '99: Proc. 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications*, pp.132–146, ACM (1999).
5) Igarashi, A., Pierce, B.C. and Wadler, P.: Featherweight Java: A minimal core calculus for Java and GJ, *ACM Trans. Program. Lang. Syst.*, Vol.23, No.3, pp.396–450 (2001).
6) Huang, S.S., Zook, D. and Smaragdakis, Y.: cJ: Enhancing java with safe type conditions, *AOSD '07: Proc. 6th international conference on Aspect-oriented software development*, New York, NY, USA, pp.185–198, ACM (2007).
7) Ligatti, J., Walker, D. and Zdancewic, S.: A type-theoretic interpretation of pointcuts and advice, *Science of Computer Programming*, Vol.63, No.3, pp.240–266 (2006).
8) Clifton, C. and Leavens, G.T.: MiniMAO: Investigating the semantics of proceed, Technical report, Department of Computer Science, Iowa State University (2005).
9) Morgan, C., Volder, K.D. and Wohlstadter, E.: A static aspect language for checking design rules, *AOSD '07: Proc. 6th international conference on Aspect-oriented software development*, New York, NY, USA, pp.63–72, ACM (2007).
10) Dantas, D.S., Walker, D., Washburn, G. and Weirich, S.: PolyAML: A polymorphic aspect-oriented functional programming language, *ICFP '05: Proc. 10th ACM SIGPLAN international conference on Functional programming*, New York, NY, USA, pp.306–319, ACM (2005).
11) Jagadeesan, R., Jeffrey, A. and Riely, J.: Typed parametric polymorphism for aspects, *Science of Computer Programming*, Vol.63, No.3, pp.267–296 (2006).
12) 青谷知幸，増原英彦：アドバイスの安全な実行のためのアスペクト指向プログラミング言語の型システム，日本ソフトウェア科学会第 24 回大会論文集 (2007).
13) Hanenberg, S. and Unland, R.: Parametric introductions, *AOSD '03: Proc. 2nd international conference on Aspect-oriented software development*, pp.80–89, ACM (2003).
14) Reppy, J. and Turon, A.: Metaprogramming with Traits, *ECOOP 2007 — Object-Oriented Programming*, Lecture Notes in Computer Science, Vol.4609, pp. 373–398 (2007).
15) Cunha, C.A., Jo a. L.S. and Monteiro, M.P.: Reusable aspect-oriented implementations of concurrency patterns and mechanisms, *AOSD '06: Proc. 5th international conference on Aspect-oriented software development*, New York, NY, USA, pp.134–145, ACM (2006).
16) Kienzle, J., Duala-Ekoko, E. and Gélineau, S.: AspectOPTIMA: A Case Study on

Aspect Dependencies and Interactions (2007).
http://www.cs.mcgill.ca/~eduala/papers/taosd-66-2006.pdf
17) 川上祐介：バイトコード変換の安全性保証—パターンマッチに基づく変換と型システム，修士論文，神戸大学大学院自然科学研究科 (2004 年度).

付　　録

A.1 Reduction Rules

健全性の証明に先立って，まず Reduction 規則の説明を行う．Reduction 規則を**図 11** に，新たな関数定義は**図 12** に示す．Reduction 操作は記号 $\rightarrow$ を用いて表す．本研究で拡張した点のみ説明する．R-INVK 規則は，呼び出すメソッドに適用されるアドバイスをルックアップ（`LookupA`）し，アドバイスを適用した結果の式へ簡約する．α は，アドバイス宣言の変数名 $\bar{x}$ とボディ e_0 の組 $(\bar{x}, e_0)$ である．`$org` 式の簡約ルール R-ORG は，適用されるアドバイスの有無にかかわらず，メソッド宣言時のボディへ簡約する．EVAL-ADVICE/INVK は，アドバイスの適用ルールである．EVAL-INVK は，適用されるアドバイスがない場合で，メソッド宣言時のボディへ簡約する．EVAL-ADVICE では，適用されるアドバイスの列から先頭の α を取り出し，そのボディへ簡約する．その際 `proceed` 変数は，残るアドバイス列 $\bar{\alpha}$ を適用した結果の式 e' に置換する．

図 12 の，$\texttt{LookupA}(m, T)$ は，クラス T のメソッド m に適用される `exec` アドバイスを探し，適用されるアドバイスの情報 α（引数名 $\bar{x}$ とボディ e の組）の列を返す．アドバイス M_i^e が適用されるかどうかは，`subsump` 関係および，$\mathcal{M}$ の元でのメソッドシグネチャの一致から判定する．`mibody` は，Method lookup と同様の方法でメソッドを探し，その引数名 $\bar{x}$ と，ボディ e の組を返す．$\texttt{mapex}_{\mathcal{M}}(e)$ は，環境 $\mathcal{M}$ による式 e の型置換操作を示す．`mibody` や `LookupA` の結果のボディ e は，$\texttt{mapex}_{\mathcal{M}}$ を施した後の式である．

A.2 健　全　性

MJ と同様の Theorem，Lemma の証明を行うが，MJ 時から変更があるもののみを記す．なお，Lemma 11〜15 は，新たに追加したものである．

Theorem 1（Subject Reduction）

$\Delta;\Gamma;\Lambda \vdash e \in T$ かつ $e \rightarrow e'$ ならば，$\Delta;\Gamma;\Lambda \vdash e' \in S$ かつ $\Delta \vdash S <: T$ なる S が存在．

Proof. $e \rightarrow e'$ の各定義について帰納的に証明．

変更した R-INVK，および新規の R-ORG，RC-ORG-RECV/ARG についてのみ示す．

Case R-INVK：e=`new` $C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d})$　`new` $C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \xrightarrow{\bar{\alpha}} e'$

reduction rules

$$\frac{\phi \vdash \texttt{fields}(C\langle\bar{T}\rangle) = \bar{U}\ \bar{f}}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).f_i \rightarrow e_i} \quad \text{(R-FLD)}$$

$$\frac{\phi;\phi \vdash \texttt{LookupA}(m, C\langle\bar{T}\rangle) = \bar{\alpha} \quad \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \xrightarrow{\bar{\alpha}} e_0}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \rightarrow e_0} \quad \text{(R-INVK)}$$

$$\frac{\phi;\phi \vdash \texttt{mibody}(m, C\langle\bar{T}\rangle) = (\bar{x}, e_0) \quad e_0' = [\bar{d}/\bar{x}, \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e})/\texttt{this}]e_0}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m\texttt{\$org}(\bar{d}) \rightarrow e_0'} \quad \text{(R-ORG)}$$

$$\frac{\phi \vdash C\langle\bar{T}\rangle <: T}{(T)\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}) \rightarrow \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e})} \quad \text{(R-CAST)}$$

$$\frac{e_0 \rightarrow e_0'}{e_0.f \rightarrow e_0'.f} \quad \text{(RC-FLD)}$$

$$\frac{e_0 \rightarrow e_0'}{e_0.m(\bar{e}) \rightarrow e_0'.m(\bar{e})} \quad \text{(RC-INV-RECV)}$$

$$\frac{e_0 \rightarrow e_0'}{e_0.m\texttt{\$org}(\bar{e}) \rightarrow e_0'.m\texttt{\$org}(\bar{e})} \quad \text{(RC-ORG-RECV)}$$

$$\frac{e_0 \rightarrow e_0'}{(T)e_0 \rightarrow (T)e_0'} \quad \text{(RC-CAST)}$$

$$\frac{e_i \rightarrow e_i'}{e_0.m(\ldots, e_i, \ldots) \rightarrow e_0'.m(\ldots, e_i', \ldots)} \quad \text{(RC-INV-ARG)}$$

$$\frac{e_i \rightarrow e_i'}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\ldots, e_i, \ldots) \rightarrow \texttt{new}\ \ C\langle\bar{T}\rangle\,(\ldots, e_i', \ldots)} \quad \text{(RC-NEW-ARG)}$$

$$\frac{e_i \rightarrow e_i'}{e_0.m\texttt{\$org}(\ldots, e_i, \ldots) \rightarrow e_0.m\texttt{\$org}(\ldots, e_i', \ldots)} \quad \text{(RC-ORG-ARG)}$$

Advice reduction

$$\frac{\phi;\phi \vdash \texttt{mibody}(m, C\langle\bar{T}\rangle) = (\bar{x}, e_0) \quad e' = [\bar{d}/\bar{x}, \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e})/\texttt{this}]e_0}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \xrightarrow{\phi} e'} \quad \text{(EVAL-INVK)}$$

$$\frac{\alpha = (\bar{x}, e_0) \quad \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \xrightarrow{\bar{\alpha}} e_0' \quad e' = [e_0'/\texttt{proceed}, \bar{d}/\bar{x}, \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e})/\texttt{this}]e}{\texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \xrightarrow{\alpha,\bar{\alpha}} e'} \quad \text{(EVAL-ADVICE)}$$

図 11　Reduction rules
Fig. 11　Reduction rules.

where $\Delta;\Lambda \vdash \texttt{LookupA}(m, C\langle\bar{T}\rangle) = \bar{\alpha}$

EVAL-INVK/ADVICE に対し，帰納的に証明する．

Case EVAL-INVK：e=`new` $C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d})$　$e' = [\bar{d}/\bar{x}, \texttt{new}\ \ C\langle\bar{T}\rangle\,(\bar{e})/\texttt{this}]e_0$

where $\Delta;\Lambda \vdash \texttt{mibody}(m, C\langle\bar{T}\rangle) = (\bar{x}, e_0)$

By T-NEW and T-INVK：`new` $C\langle\bar{T}\rangle\,(\bar{e}).m(\bar{d}) \in U_0$　`new` $C\langle\bar{T}\rangle\,(\bar{e}) \in C\langle\bar{T}\rangle$

$\Delta;\Lambda \vdash \texttt{mitype}(m, C\langle\bar{T}\rangle) = \bar{U} \rightarrow U_0$　$\Delta;\Gamma;\Lambda \vdash \bar{d} \in \bar{V}$　$\Delta \vdash \bar{V} <: \bar{U}$

Advice lookup

$$\bar{\alpha}=\left\{(\bar{x},e')\;\middle|\;\begin{array}{l}\Delta;\Lambda\vdash \mathtt{mtype}(m,C\!<\!\bar{T}\!>)=\bar{U}\to U_0 \quad \mathtt{aspect}\,A\{\cdots\bar{M}^e\}\\ M_i^e\in\bar{M}^e \quad \mathtt{aInfo}(M_i^e)=(\Delta',\Gamma',e,\bar{S}\to S_0,T',\langle\Lambda',m',S'\rangle)\\ \Delta',\Delta;\mathcal{M}\vdash\mathtt{subsump}(T',\langle\Lambda',m',S'\rangle,C\!<\!\bar{T}\!>,\langle\Lambda,m,S'\rangle)\\ \bar{U}=\mathtt{mapT}_{\mathcal{M}}(\bar{S}) \quad U_0=\mathtt{mapT}_{\mathcal{M}}(S_0) \quad \bar{x}\mapsto\bar{S}\subset\Gamma'\\ \mathtt{dropPre}(\langle\Lambda',m'\rangle,\langle\Lambda,m\rangle)=m_1':m_1 \quad e'=[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}(e)\end{array}\right\}$$

$$\Delta;\Lambda\vdash\mathtt{LookupA}(m,C\!<\!\bar{T}\!>)=\bar{\alpha}$$

(Advice-Lookup)

Method Body lookup

$$\frac{\Delta;\Lambda\vdash\mathtt{mbody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e) \quad or\ \ \Delta;\Lambda\vdash\mathtt{ibody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e)}{\Delta;\Lambda\vdash\mathtt{mibody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e)}$$

(MB)

$$\frac{\Delta;\Lambda\not\vdash\mathtt{mbody}(m,C\!<\!\bar{T}\!>) \quad \Delta;\Lambda\not\vdash\mathtt{ibody}(m,C\!<\!\bar{T}\!>) \quad CT(C)=\mathtt{class}\ C\!<\!\bar{X}\!\lhd\ \bar{N}\!>\lhd T\{\cdots\} \quad \Delta;\Lambda\vdash\mathtt{mibody}(m,[\bar{T}/\bar{X}]T)=(\bar{x},e)}{\Delta;\Lambda\vdash\mathtt{mibody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e)}$$

(MB-SUP)

$$\frac{\begin{array}{c}CT(C)=\mathtt{class}\ C\!<\!\bar{X}\!\lhd\ \bar{N}\!>\lhd T\{\cdots\bar{M}\} \quad M_i\in\bar{M}\\ \mathtt{mInfo}(M_i,C\!<\!\bar{X}\!\lhd\bar{N}\!>)=(\Delta',\Gamma,e,\bar{S}\to S_0,\langle\Lambda',m',S'\rangle)\\ \Delta_k=[\bar{T}/\bar{X}]\Delta',\Delta \quad \Lambda_k=[\bar{T}/\bar{X}]\Lambda' \quad S_k=[\bar{T}/\bar{X}]S'\\ \Delta_k;\mathcal{M}\vdash\mathtt{subsump}(C\!<\!\bar{T}\!>,\langle\Lambda_k,m',S_k\rangle,C\!<\!\bar{T}\!>,\langle\Lambda,m,S_k\rangle)\\ \bar{x}\mapsto\bar{S}\subset\Gamma \quad \mathtt{dropPre}(\langle\Lambda',m'\rangle,\langle\Lambda,m\rangle)=m_1':m_1 \quad e'=[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}([\bar{T}/\bar{X}]e)\end{array}}{\Delta;\Lambda\vdash\mathtt{mbody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e')}$$

(MT-B)

$$\frac{\begin{array}{c}\mathtt{aspect}\ A\{\cdots\bar{M}^a\} \quad M_i^a\in\bar{M}^a\\ \mathtt{aInfo}(M_i^a)=(\Delta',\Gamma,e,\bar{S}\to S_0,T',\langle\Lambda',m',S'\rangle)\\ \Delta',\Delta;\mathcal{M}\vdash\mathtt{subsump}(T',\langle\Lambda',m',S'\rangle,T,\langle\Lambda,m,S'\rangle)\\ \bar{x}\mapsto\bar{S}\subset\Gamma \quad \mathtt{dropPre}(\langle\Lambda',m'\rangle,\langle\Lambda,m\rangle)=m_1':m_1 \quad e'=[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}(e)\end{array}}{\Delta;\Lambda\vdash\mathtt{ibody}(m,T)=(\bar{x},e')}$$

(IT-B)

Type Mapping to Expressions

$$\mathtt{mapex}_{\mathcal{M}}(x)=x$$

(MAP-VAR)

$$\frac{\mathtt{mapex}_{\mathcal{M}}(e)=e'}{\mathtt{mapex}_{\mathcal{M}}(e.f)=e'.f}$$

(MAP-FLD)

$$\frac{\mathtt{mapex}_{\mathcal{M}}(e)=e' \quad \mathtt{mapex}_{\mathcal{M}}(\bar{e})=\bar{e}'}{\mathtt{mapex}_{\mathcal{M}}(e.m(\bar{e}))=e'.m(\bar{e}')}$$

(MAP-INV)

$$\frac{\mathtt{mapT}_{\mathcal{M}}(\bar{T})=\bar{T}' \quad \mathtt{mapex}_{\mathcal{M}}(\bar{e})=\bar{e}'}{\mathtt{mapex}_{\mathcal{M}}(\mathtt{new}\,C\!<\!\bar{T}\!>(\bar{e}))=\mathtt{new}\,C\!<\!\bar{T}'\!>(\bar{e}')}$$

(MAP-NEW)

$$\frac{\mathtt{mapT}_{\mathcal{M}}(T)=T' \quad \mathtt{mapex}_{\mathcal{M}}(e)=e'}{\mathtt{mapex}_{\mathcal{M}}((T)e)=(T')e'}$$

(MAP-CAST)

$$\frac{\mathtt{mapex}_{\mathcal{M}}(e)=e' \quad \mathtt{mapex}_{\mathcal{M}}(\bar{e})=\bar{e}'}{\mathtt{mapex}_{\mathcal{M}}(e.m\$\mathtt{org}(\bar{e}))=e'.m\$\mathtt{org}(\bar{e}')}$$

(MAP-ORG)

図 12　Reduction sub rules
Fig. 12　Reduction sub rules.

mitype と mibody は対になっており，mitype が成立すれば，対応する mibody の規則も成立する．mitype の各規則に対し，それぞれ証明を行う．

Case　MT-CLS：$\mathtt{mtype}(m,C\!<\!\bar{T}\!>)=\bar{U}\to U_0$

$CT(C)=\mathtt{class}\ C\!<\!\bar{X}\!\lhd\ \bar{N}\!>\lhd T\{\cdots\bar{M}\}$　$\mathtt{mInfo}(M_i,C\!<\!\bar{X}\!\lhd\bar{N}\!>)=(\Delta',\Gamma',e_0',\bar{S}\to S_0,\langle\Lambda',m',S'\rangle)$

$\Delta_k=\Delta,[\bar{T}/\bar{X}](\Delta')$　$\Lambda_k=[\bar{T}/\bar{X}](\Lambda')$　$S_k=[\bar{T}/\bar{X}]S$

$\Delta_k;\mathcal{M}\vdash\mathtt{subsump}(C\!<\!\bar{T}\!>,\langle\Lambda_k,m',S_k\rangle,C\!<\!\bar{T}\!>,\langle\Lambda,m,S_k\rangle)$

$\bar{U}=[\bar{T}/\bar{X}]\mathtt{mapT}_{\mathcal{M}}(\bar{S})$　$U_0=[\bar{T}/\bar{X}]\mathtt{mapT}_{\mathcal{M}}(S_0)$　$\mathtt{dropPre}(\langle\Lambda',m'\rangle,\langle\Lambda,m\rangle)=m_1':m_1$

By mbody：$\Delta;\Lambda\vdash\mathtt{mbody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e_0)$ where $e_0=[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}([\bar{T}/\bar{X}]e_0')$

$C\!<\!\bar{T}\!>$ に対する mbody の結果，e_0 は M_i のボディ e_0' を型置換，$\mathcal{M}$ 変換，名前置換したものとなっている．

By T-MTD：$\bar{X}\!<\!:\bar{N},\Delta';\Gamma';\Lambda'\vdash e_0'\in S_0'$　$\bar{X}\!<\!:\bar{N},\Delta'\vdash S_0'\!<\!:S_0$ と，e_0' は型付けされている．

Lemma 15（式の型置換），12（式の $\mathcal{M}$ 変換），14（名前置換）を適用し，

$\Delta;\mathtt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}]\Gamma');\Lambda\vdash[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}([\bar{T}/\bar{X}]e_0')\in W_0$　$\Delta\vdash W_0\!<\!:[\bar{T}/\bar{X}]\mathtt{mapT}_{\mathcal{M}}(S_0')$ と型付け可能．

このとき，Lemma 7（型置換のサブタイプ保存），11（$\mathcal{M}$ 変換のサブタイプ保存）より，$\Delta\vdash W_0<:\ [\bar{T}/\bar{X}]\mathtt{mapT}_{\mathcal{M}}(S_0')<:\ [\bar{T}/\bar{X}]\mathtt{mapT}_{\mathcal{M}}(S_0)=U_0$.

引数 $\bar{d}$ で呼び出した結果は，Lemma 5（項置換）より，

$\Delta;\Gamma;\Lambda\vdash[\bar{d}/\bar{x},\ \mathtt{new}\ C\!<\!\bar{T}\!>(\bar{e})/\mathtt{this}]e_0\in W_0'$　$\Delta\vdash W_0'\!<\!:W_0\!<\!:U_0$

Case　IT：$\mathtt{itype}(m,C\!<\!\bar{T}\!>)=\bar{U}\to U_0$

$\mathtt{aspect}\ A\{\cdots\bar{M}^a\}$　$\mathtt{aInfo}(M_i^a)=(\Delta',\Gamma',e_0',\bar{S}\to S_0,T',\langle\Lambda',m',S'\rangle)$

$\Delta',\Delta;\mathcal{M}\vdash\mathtt{subsump}(T',\langle\Lambda',m',S'\rangle,T,\langle\Lambda,m,S'\rangle)$

$U_0=\mathtt{mapT}_{\mathcal{M}}(S_0)$　$\bar{U}=\mathtt{mapT}_{\mathcal{M}}(\bar{S})$　$\mathtt{dropPre}(\langle\Lambda',m'\rangle,\langle\Lambda,m\rangle)=m_1':m_1$

By ibody：$\Delta;\Lambda\vdash\mathtt{ibody}(m,C\!<\!\bar{T}\!>)=(\bar{x},e_0)$ where $e_0=[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}(e_0')$

$C\!<\!\bar{T}\!>$ に対する ibody の結果，e_0 は M_i^a のボディ e_0' を $\mathcal{M}$ 変換，名前置換したものとなっている．

By T-ADD：$\Delta';\Gamma';\Lambda'\vdash e_0'\in S_0'$　$\Delta'\vdash S_0'\!<\!:S_0$ と，e_0' は型付けされている．

Lemma 12（式の $\mathcal{M}$ 変換），14（名前置換）を適用し，

$\Delta;\mathtt{mapT}_{\mathcal{M}}(\Gamma');\Lambda\vdash[m_1/m_1']\mathtt{mapex}_{\mathcal{M}}(e_0')\in W_0$　$\Delta\vdash W_0\!<\!:\mathtt{mapT}_{\mathcal{M}}(S_0')$ と，型付け可能．

このとき Lemma 11（$\mathcal{M}$ 変換のサブタイプ保存）より，$\Delta\vdash W_0\!<\!:\mathtt{mapT}_{\mathcal{M}}(S_0')\!<\!:\mathtt{mapT}_{\mathcal{M}}(S_0)=U_0$.

引数 $\bar{d}$ で呼び出した結果は，Lemma 5（項置換）より，

$\Delta;\Gamma;\Lambda\vdash[\bar{d}/\bar{x},\ \mathtt{new}\ C\!<\!\bar{T}\!>(\bar{e})/\mathtt{this}]e_0\in W_0'$　$\Delta\vdash W_0'\!<\!:W_0\!<\!:U_0$

Case　MT-SUP，IT-SUP：自明である．

Case EVAL-ADVICE：e=new $C\langle\bar{T}\rangle(\bar{e}).m(\bar{d})$　new $C\langle\bar{T}\rangle(\bar{e}).m(\bar{d}) \xrightarrow{\bar{\alpha}} e_0''$　$\alpha=(\bar{x}, e_0)$

$e'=[e_0''/\texttt{proceed}, \bar{d}/\bar{x}, \texttt{new}\ C\langle\bar{T}\rangle(\bar{e})/\texttt{this}]e_0$ where $\Delta;\Lambda\vdash\texttt{LookupA}(m, C\langle\bar{T}\rangle)=\bar{\alpha}'$　$\alpha{:}\bar{\alpha}\subset\bar{\alpha}'$

By T-NEW and T-INVK：new $C\langle\bar{T}\rangle(\bar{e}).m(\bar{d})\in U_0$　new $C\langle\bar{T}\rangle(\bar{e})\in C\langle\bar{T}\rangle$

$\Delta;\Gamma;\Lambda\vdash\bar{d}\in\bar{V}$　$\Delta;\Lambda\vdash\texttt{mitype}(m, C\langle\bar{T}\rangle)=\bar{U}\to U_0$　$\Delta\vdash\bar{V}<:\bar{U}$

帰納法の仮定より，$\Delta;\Gamma;\Lambda\vdash e_0''\in W_0$　$\Delta\vdash W_0<:U_0$

By LookupA：$\alpha=(\bar{x}, e_0)\in\bar{\alpha}'$　$e_0=[m_1/m_1']\texttt{mapex}_{\mathcal{M}}(e_0')$

$\texttt{aInfo}(M_i^e)=(\Delta', \Gamma', e_0', \bar{S}\to S_0, T', \langle\Lambda', m', S'\rangle)$

$\Delta', \Delta;\mathcal{M}\vdash\texttt{subsump}(T', \langle\Lambda', m', S'\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S'\rangle)$

$U_0=\texttt{mapT}_{\mathcal{M}}(S_0)$　$\bar{U}=\texttt{mapT}_{\mathcal{M}}(\bar{S})$　$\texttt{dropPre}(\langle\Lambda', m'\rangle, \langle\Lambda, m\rangle)=m_1':m_1$

適用するアドバイスのボディe_0は，M_i^e のボディe_0'を $\mathcal{M}$ 変換，名前置換したものとなる．

By T-EXEC：$\Delta';\Gamma', \texttt{proceed}\mapsto S_0;\Lambda'\vdash e_0'\in S_0'$　$\Delta'\vdash S_0'<:S_0$ と，e_0' は型付けされている．

Lemma 12（式の $\mathcal{M}$ 変換），14（名前置換）を適用し，

$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma'), \texttt{proceed}\mapsto U_0;\Lambda\vdash[m_1/m_1']\texttt{mapex}_{\mathcal{M}}(e_0')\in W_0$　$\Delta\vdash W_0<:\texttt{mapT}_{\mathcal{M}}(S_0')$ と，型付け可能．

このとき Lemma 11（$\mathcal{M}$ 変換のサブタイプ保存）より，$\Delta\vdash W_0<:\texttt{mapT}_{\mathcal{M}}(S_0')<:\texttt{mapT}_{\mathcal{M}}(S_0)=U_0$.

引数 $\bar{d}$ で呼び出した結果は，Lemma 5（項置換）より，

$\Delta;\Gamma;\Lambda\vdash[e_0''/\texttt{proceed}, \bar{d}/\bar{x}, \texttt{new}\ C\langle\bar{T}\rangle(\bar{e})/\texttt{this}]e_0\in W_0'$　$\Delta\vdash W_0'<:W_0<:U_0$

Case R-ORG：$e=$ new $C\langle\bar{T}\rangle(\bar{e}).m\$\texttt{org}(\bar{d})$　$e'=[\bar{d}/\bar{x}, \texttt{new}\ C\langle\bar{T}\rangle(\bar{e})/\texttt{this}]e_0$

where $\Delta;\Lambda\vdash\texttt{mibody}(m, C\langle\bar{T}\rangle)=(\bar{x}, e_0)$

前述の EVAL-INVK と同様に証明可能．

Case RC-ORG-RECV/ARG：自明である．

Theorem 2（Progress）

$\phi;\phi;\phi\vdash e\in T$ なる e が，well-typed なとき，

1) 式 e がその中に new $C\langle\bar{T}\rangle(\bar{e}).f$ を持つとき，$\texttt{fields}(C\langle\bar{T}\rangle)=\bar{U}\ \bar{f}$　$f=f_i$.

2) 式 e がその中に new $C\langle\bar{T}\rangle(\bar{e}).m(\bar{d})$ を持つとき，$\phi;\phi\vdash\texttt{mibody}(m, C\langle\bar{T}\rangle)=(\bar{x}, e_0)$　$\#(\bar{x})=\#(\bar{d})$.

3) 式 e がその中に new $C\langle\bar{T}\rangle(\bar{e}).m\$\texttt{org}(\bar{d})$ を持つとき，$\phi;\phi\vdash\texttt{mibody}(m, C\langle\bar{T}\rangle)=(\bar{x}, e_0)$　$\#(\bar{x})=\#(\bar{d})$.

4) 式 e がその中に new $C\langle\bar{T}\rangle(\bar{e}).m(\bar{d})$ を持つとき，

$\phi;\phi\vdash\texttt{LookupA}(m, C\langle\bar{T}\rangle)=\bar{\alpha}$　$\forall\alpha_i\in\bar{\alpha}$，$\alpha_i=(\bar{x}, e_0)$　$\#(\bar{x})=\#(\bar{d})$.

Proof. 1)，2) は MJ 時と同様なので省略する．3) についても 2) と同様に証明可能．4) は LookupA で mtype の結果と M_i^e のシグネチャの一致を調べることから自明である．

Theorem 3（Type Soundness）

$\phi;\phi;\phi\ \vdash e\in T$ なる e を，$e\to^* e'$ によりノーマルフォームへ簡約した e' は，変数 v $(\phi;\phi;\phi\vdash v\in S, \phi\vdash S<:T)$，または (T)new $C\langle\bar{T}\rangle(\bar{e})$ を含んだ式 $(\phi\vdash T<:C\langle\bar{T}\rangle)$ のいずれかになる．

Proof. Theorem 1，Theorem 2 から，即座に求まる．

Lemma 1（Name Uniqueness）

Lemma 1.1：class $C\langle\bar{X}\triangleleft\bar{N}\rangle\triangleleft T\{\cdots\bar{M}\}$ ok において，$\Delta;\Lambda\vdash\texttt{mtype}(m, C\langle\bar{T}\rangle)=\bar{U}\to U_0$ を成立させる M_i（$\in\bar{M}$）は 2 つ以上は存在しない．

Proof. $\texttt{mtype}(m, C\langle\bar{T}\rangle)$ が MT-SUP により求まる場合は簡単で，$\Delta;\Lambda\vdash\texttt{mtype}(m, C\langle\bar{T}\rangle)=\bar{U}\to U_0$ を成立させる M_i が存在しない（0 個）．

$\texttt{mtype}(m, C\langle\bar{T}\rangle)$ が MT-CLS により求まる場合は，2 つの M_1，$M_2\in\bar{M}$ で MT-CLS が成立すると仮定し，矛盾が起こることを示す．

$\texttt{mInfo}(M_1, C\langle\bar{X}\triangleleft\bar{N}\rangle)=(\Delta_1, \_, \_, \_, \langle\Lambda_1, m_1, S_1\rangle)$

$\texttt{mInfo}(M_2, C\langle\bar{X}\triangleleft\bar{N}\rangle)=(\Delta_2, \_, \_, \_, \langle\Lambda_2, m_2, S_2\rangle)$

$\Delta_1'=\Delta, [\bar{T}/\bar{X}](\Delta_1)$　$\Lambda_1'=[\bar{T}/\bar{X}](\Lambda_1)$　$S_1'=[\bar{T}/\bar{X}]S_1$

$\Delta_2'=\Delta, [\bar{T}/\bar{X}](\Delta_2)$　$\Lambda_2'=[\bar{T}/\bar{X}](\Lambda_2)$　$S_2'=[\bar{T}/\bar{X}]S_2$

$\Delta_1';\mathcal{M}_1\vdash\texttt{subsump}(C\langle\bar{T}\rangle, \langle\Lambda_1, m_1, S_1'\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S_1'\rangle)$

$\Delta_2';\mathcal{M}_2\vdash\texttt{subsump}(C\langle\bar{T}\rangle, \langle\Lambda_2, m_2, S_2'\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S_2'\rangle)$

一方で，T-CLS より，M_1，M_2 間に，$\Delta,\Delta_1,\Delta_2\vdash\texttt{disjoint}(\langle\Lambda_1, m_1, S_1\rangle, \langle\Lambda_2, m_2, S_2\rangle)$ が成立．

disjoint がその各規則で成立する場合ごとに，矛盾を導く．

Case DS-STATIC：$\Lambda_1\vdash\texttt{static}(m_1)$，$\Lambda_2\vdash\texttt{static}(m_2)$，$m_1\neq m_2$

上記仮定の subsump に関し，sump-S 規則により，$m=m_1=m_2$ となり，矛盾が起きる．

Case DS-UNIQ：$m_1=\overline{pre1\#}n_1$，$m_2=\overline{pre2\#}n_2$，$pre1_i\neq pre2_i$

sump-R 規則内の dropPre 定義より，$\langle\Lambda_1, m_1\rangle\sqsupseteq_{id}\langle\Lambda, m\rangle$，$\langle\Lambda_2, m_2\rangle\sqsupseteq_{id}\langle\Lambda, m\rangle$

$\sqsupseteq_{id}$の定義より，$m=\overline{pre\#}n$ のとき，$\forall i\ pre_i=pre1_i=pre2_i$ となり，矛盾が起きる．

Case DS-TYPE：$S_1'<:S_2'$ *or* $S_2'<:S_1'$　$\Lambda_1\not\vdash\texttt{static}(m_1)$　$\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda_2, m_2\rangle)=m_1':m_2'$

$\Delta;\Lambda_1\vdash\texttt{mtype}(m_1', S_1')=\bar{U}\to U_0$　$\Delta;\Lambda_2\vdash\texttt{mtype}(m_2', S_2')=\bar{V}\to V_0$

$\bar{Y}=\texttt{pmVars}(\Lambda_1), \texttt{pmVars}(\Lambda_2)$　$\Delta;\bar{Y}\not\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$

Case $\Lambda_2\vdash\texttt{static}(m_2)$ のとき：

(m_2, m) 間の subsump と sump-S 定義より $m_2=m$

および，Lemma 9（サブタイプにおける mitype 保存）により，S_1'，S_2' 間で mitype が保存され，$\texttt{mtype}(m_2', S_1')=\texttt{mtype}(m_2', S_2')=\bar{V}\to V_0$ である．

(m_1, m) 間の subsump と sump-R の定義より，

$\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda, m\rangle)=m_1':m'$　$\Delta;\Lambda\vdash\texttt{mtype}(m_1', S_1')=\bar{U}\to U_0$　$\Delta;\Lambda\vdash\texttt{mtype}(m', S_1')=\bar{V}\to V_0$

$\Delta;\mathcal{M};\bar{Y}\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$ となる．

これは，DS-TYPE の条件である $\Delta;\bar{Y}\not\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$ と矛盾する．

Case　$\Lambda_2\not\vdash\texttt{static}(m_2)$ のとき：

Case　m_1，m_2 に対する m の dropPre の結果が，同一の m' になる場合：

$\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda, m\rangle)=m_1':m'$　$\texttt{dropPre}(\langle\Lambda_2, m_2\rangle, \langle\Lambda, m\rangle)=m_2':m'$

sump-R の定義より，$\Delta;\Lambda\vdash\texttt{mtype}(m', S_1')=\bar{W}\to W_0$

$\bar{Y}_1=\texttt{pmVars}(\Lambda_1)$　$\Delta;\mathcal{M}_1;\bar{Y}_1\vdash\texttt{tunify}(U_0{:}\bar{U}, W_0{:}\bar{W})$

$\bar{Y}_2=\texttt{pmVars}(\Lambda_2)$　$\Delta;\mathcal{M}_2;\bar{Y}_2\vdash\texttt{tunify}(V_0{:}\bar{V}, W_0{:}\bar{W})$

プログラムの前提より，型変数はグローバルに一意なので，$\bar{Y}_1\cap\bar{Y}_2=\phi$．

Lemma 2 より，$\Delta;\mathcal{M};\bar{Y}_1,\bar{Y}_2\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$ なる $\mathcal{M}$ が存在し，矛盾．

Case　m_1，m_2 に対する m の dropPre の結果が異なる場合：

m，m_1，m_2 の間での dropPre および $\sqsupseteq_{id}$ が成立するのは，以下の状況に限定される．

$m_1=u_1$　$m_2=pre\#u_2$　$m=pre\#m'$　（u は名前変数を表す）

$\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda, m\rangle)=u_1:pre\#m'$　$\texttt{dropPre}(\langle\Lambda_2, m_2\rangle, \langle\Lambda, m\rangle)=u_2:m'$

(m_1, m)，(m_2, m) の subsump 関係から，$\texttt{dom}(pre\#m')\subset\texttt{dom}(u_1)$　$\texttt{dom}(pre\#m')\subset\texttt{dom}(pre\#u_2)$

一方，(m_1, m_2) 間の disjoint 成立より，$\texttt{dom}(u_1)\cap\texttt{dom}(pre\#u_2)=\phi$ となり矛盾．

Case　DS-RANGE：m_1 を reflective な名前，m_2 を static な名前とする．

$\texttt{dropPre}(\langle\Lambda_1, m_1\rangle, \langle\Lambda_2, m_2\rangle)=m_1':m_2'$　$\Delta;\Lambda_1\vdash\texttt{mtype}(m_1', S_1')=\bar{U}\to U_0$　$\Delta;\Lambda_2\vdash\texttt{mtype}(m_2', S_1')=\bar{V}\to V_0$

$\bar{Y}=\texttt{pmVars}(\Lambda_1), \texttt{pmVars}(\Lambda_2)$　$\Delta;\bar{Y}\not\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$

(m_2, m) 間の subsump と sump-S 定義より，$m_2=m$

(m_1, m) 間の subsump と sump-R 定義より，$\Delta_1';\mathcal{M}_1\vdash\texttt{subsump}(C{<}\bar{T}{>}, \langle\Lambda_1, m_1, S_1'\rangle, C{<}\bar{T}{>}, \langle\Lambda, m, S_1'\rangle)$

より $\Delta;\mathcal{M};\bar{Y}\vdash\texttt{tunify}(U_0{:}\bar{U}, V_0{:}\bar{V})$ が成立し，矛盾．

Lemma 1.2： aspect $A\{\cdots\bar{M}^a\}$ ok において，$\Delta;\Lambda\vdash\texttt{itype}(m, C{<}\bar{T}{>})=\bar{U}\to U_0$ を成立させる $M_i^a(\in\bar{M}^a)$ は 2 つ以上は存在しない．

Proof. $\texttt{itype}(m, C{<}\bar{T}{>})$ が IT-SUP により求まる場合は簡単で，$\Delta;\Lambda\vdash\texttt{itype}(m, C{<}\bar{T}{>})=\bar{U}\to U_0$ を成立させる M_i^a が存在しない（0 個）．

$\texttt{itype}(m, C{<}\bar{T}{>})$ が IT により求まる場合は，2 つの M_1^a，$M_2^a\in\bar{M}^a$ で IT が成立すると仮定し，矛盾が起こることを示す．

$\texttt{aInfo}(M_1^a)=(\Delta_1, \_, \_, \_, T_1, \langle\Lambda_1, m_1, S_1\rangle)$　$\texttt{aInfo}(M_2^a)=(\Delta_2, \_, \_, \_, T_2, \langle\Lambda_2, m_2, S_2\rangle)$

$\Delta_1'=\Delta,\Delta_1$　$\Delta_1';\mathcal{M}_1\vdash\texttt{subsump}(T_1, \langle\Lambda_1, m_1, S_1'\rangle, C{<}\bar{T}{>}, \langle\Lambda, m, S_1'\rangle)$

$\Delta_2'=\Delta,\Delta_2$　$\Delta_2';\mathcal{M}_2\vdash\texttt{subsump}(T_2, \langle\Lambda_2, m_2, S_2'\rangle, C{<}\bar{T}{>}, \langle\Lambda, m, S_2'\rangle)$

By T-ASPECT：$\Delta_1', \Delta_2'\vdash\texttt{disconf}(T_1, \langle\Lambda_1, m_1, S_1\rangle, T_2, \langle\Lambda_2, m_2, S_2\rangle)$

disconf がその各規則で成立する場合ごとに矛盾を導く．

Case　DSC-CLS：$no\,\mathcal{M}$　$\bar{Z}=\texttt{Var}_\Delta(\Lambda_1, T_1), \texttt{Var}_\Delta(\Lambda_2, T_2)$　$\forall Z_i\in\bar{Z}$，　$Z_i\mapsto T\in\mathcal{M}$

$\Delta;\bar{Y}\vdash T\prec:Z_i$　$\Delta\vdash\texttt{mapT}_\mathcal{M}(T_1){<:}\texttt{mapT}_\mathcal{M}(T_2)$ *or* $\Delta\vdash\texttt{mapT}_\mathcal{M}(T_2){<:}\texttt{mapT}_\mathcal{M}(T_1)$

一方，仮定の subsump 関係の成立より，

$\bar{Z}_1=\texttt{Var}_\Delta(\Lambda_1, T_1)$　$\Delta;\mathcal{M}_1';\bar{Z}_1\vdash\texttt{tunify}(T_1, C{<}\bar{T}{>})$

$\bar{Z}_2=\texttt{Var}_\Delta(\Lambda_2, T_2)$　$\Delta;\mathcal{M}_2';\bar{Z}_2\vdash\texttt{tunify}(T_2, C{<}\bar{T}{>})$

プログラムの前提より，型変数はグローバルに一意なので，$\bar{Z}_1\cap\bar{Z}_2=\phi$．

Lemma 2 より，$\Delta;\mathcal{M};\bar{Z}_1,\bar{Z}_2\vdash\texttt{tunify}(T_1, T_2)$ なる $\mathcal{M}$ が存在し，矛盾．

Case　DSC-NAME：$\Delta\vdash\texttt{disjoint}(\langle\Lambda_1, m_1, S_1\rangle, \langle\Lambda_2, m_2, S_2\rangle)$ Lemma 1.1 と同様に証明可能．

Case　DSC-RANGE：$\bar{Z}=\texttt{Var}_\Delta(\Lambda_1, T_1), \texttt{Var}_\Delta(\Lambda_1, T_2)$　$\Delta;\bar{Z}\not\vdash\texttt{tunify}(T_1, T_2)$

一方，仮定の subsump 関係成立と Lemma 2 より，

$\bar{Z}_1=\texttt{Var}_\Delta(\Lambda_1, T_1)$　$\Delta;\mathcal{M}_1';\bar{Z}_1\vdash\texttt{tunify}(T_1, C{<}\bar{T}{>})$

$\bar{Z}_2=\texttt{Var}_\Delta(\Lambda_2, T_2)$　$\Delta;\mathcal{M}_2';\bar{Z}_2\vdash\texttt{tunify}(T_2, C{<}\bar{T}{>})$

$\Delta;\mathcal{M};\bar{Z}\vdash\texttt{tunify}(T_1, T_2)$ となる $\mathcal{M}$ が存在し，矛盾．

Lemma 1.3： $\Delta;\Lambda\vdash\texttt{mitype}(m, C{<}\bar{T}{>})=\bar{U}\to U_0$ において，$\Delta;\Lambda\vdash\texttt{itype}(m, C{<}\bar{T}{>})$ と $\Delta;\Lambda\vdash\texttt{mtype}(m, C{<}\bar{T}{>})$ は同時には成立しない．

Proof. 構文上の前提において，匿名機構を用いてクラス宣言とアスペクト宣限でメソッド名を排他としているため，itype と mtype は同時に成立しない．

Lemma 2（Type Unification）

$\Delta;\mathcal{M}_1;\bar{Y}\vdash\texttt{tunify}(\bar{U}, \bar{T})$，$\Delta;\mathcal{M}_2;\bar{Z}\vdash\texttt{tunify}(\bar{V}, \bar{T})$ かつ，$\bar{Y}\cap\bar{Z}=\phi$ であれば，$\Delta;\mathcal{M}_3;\bar{Y},\bar{Z}\vdash\texttt{tunify}(\bar{U}, \bar{V})$ なる $\mathcal{M}_3$ が存在．

Proof. MJ 論文での Lemma 2 に，条件として $\bar{Y}\cap\bar{Z}=\phi$ を追加している．証明については，関連する **Lemma 3**，**4** ともども，論文 2) 参照．

Lemma 5（TERM SUBSTITUTION PRESERVES TYPING）

非型変数をバウンドするいかなる Δ においても，$\Delta;\Gamma,\bar{x}\mapsto\bar{T};\Lambda\vdash e\in T_0$ かつ $\Delta;\Gamma;\Lambda\vdash\bar{d}\in\bar{S}$ $(\Delta\vdash\bar{S}{<:}\bar{T})$ ならば，$\Delta;\Gamma;\Lambda\vdash[\bar{d}/\bar{x}]e\in S_0$ と型付け可能で，このとき $\Delta\vdash S_0{<:}T_0$ である．

Proof. 証明は帰納的に行われ，$org 式については T-INVK 同様に証明可能．

Lemma 6（NARROWING）

1) $\Delta, X{<:}S \vdash T_1{<:}T_2$ かつ $\Delta \vdash U{<:}S$ ならば，$\Delta, X{<:}U \vdash T_1{<:}T_2$.

2) $\Delta, S{<:}X \vdash T_1{<:}T_2$ かつ $\Delta \vdash S{<:}U$ ならば，$\Delta, U{<:}X \vdash T_1{<:}T_2$.

Lemma 7（TYPE SUBSTITUTION PRESERVES SUBTYPING）

$\Delta_1, \bar{X}{<:}\bar{N}$, $\Delta_2 \vdash S{<:}T$, $\Delta_1 \vdash \bar{U}{<:}[\bar{U}/\bar{X}]\bar{N}$，かつ $\Delta_1 \vdash \bar{U}$ `ok` かつ $\bar{X}$ が Δ_1 に現れないならば，$\Delta_1, [\bar{U}/\bar{X}]\Delta_2 \vdash [\bar{U}/\bar{X}]S{<:}[\bar{U}/\bar{X}]T$

Lemma 8

$\Delta \vdash S{<:}T$ かつ $\texttt{fields}(\texttt{bound}_\Delta(T)) = \bar{U}\ \bar{f}$ ならば，

$\texttt{fields}(\texttt{bound}_\Delta(S)) = \cdots, \bar{V}\ \bar{f}$ かつ $\Delta \vdash V_i{<:}U_i$ が成立.

Lemma 9

$\Delta \vdash T$ `ok` かつ $\Delta;\Lambda \vdash \texttt{mitype}(m, T) = \bar{U} \to U_0$ ならば，

$\Delta \vdash S{<:}T$ かつ $\Delta \vdash S$ `ok` なる S に対し，$\Delta;\Lambda \vdash \texttt{mitype}(m, \texttt{bound}_\Delta(S)) = \bar{U} \to U_0$ が成立.

Proof. サブタイプ規則 $\Delta \vdash S{<:}T$ のそれぞれについて帰納的に証明.

S-REFL/TRANS/VAR の Case に変更はないので，S-CLS についてのみ記す.

By S-CLS：$\Delta \vdash C\langle\bar{T}\rangle <: [\bar{T}/\bar{X}]T$

$\texttt{mitype}(m, C\langle\bar{T}\rangle)$ が MT/IT-SUP ルールを用いて得られる場合の証明は，自明なので省略.

MT-Λ，MT-CLS，IT で成立する場合ごとに証明する.

Case MT-Λ：$\Lambda = C\langle\bar{T}\rangle \mapsto \langle \_, m, U_0\ m(\bar{U})\rangle$

各 reflective ブロックで宣言された名前変数はグローバルに一意なものなので，$\Delta;\Lambda \vdash \texttt{mtype}(m, [\bar{T}/\bar{X}]T)$ も MT-Λ で成立しなければならない．しかし，Λ は 1 個のマッピングしか持たないため，$\Delta;\Lambda \vdash \texttt{mitype}(m, [\bar{T}/\bar{X}]T)$ は成立せず，前提と矛盾する．よって，$\Delta;\Lambda \vdash \texttt{mitype}(m, C\langle\bar{T}\rangle)$ は MT-Λ 以外で成立しているはずである.

Case MT-CLS：$CT(C) = \texttt{class}\ C\langle\bar{X} \triangleleft \bar{N}\rangle \triangleleft T\{\cdots\}$　$\texttt{mInfo}(M_i, C\langle\bar{X}\rangle) = (\Delta_i, \_, \_, \_, \langle\Lambda', m_i, S_i\rangle)$

$\Delta; \mathcal{M}_i \vdash \texttt{subsump}(C\langle\bar{T}\rangle, \langle\Lambda_i, m_i, S_i\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S\rangle)$

By T-MTD：$\Delta_i, \Delta; \Lambda \vdash \texttt{override}(m_i, T, \bar{U} \to U_0)$ `ok in` $C\langle\bar{X}\rangle$

override がその各規則で成立する場合ごとに証明.

Case OVR-SUCCESS : $C\langle\bar{T}\rangle$ のメソッド m（シグネチャ $\bar{U} \to U_0$）において，

$\Delta;\Lambda \vdash \texttt{mtype}(m, [\bar{T}/\bar{X}]T) = \bar{V} \to V_0$　$\bar{U} = \bar{V}$　$U_0 = V_0$ が成立している.

Case OVR-INSTANT：$\texttt{inst}(T)$　$\Delta;\Lambda \not\vdash \texttt{mtype}(m, T)$

`inst` の定義より，T のメンバ一覧は $C\langle\bar{X}\rangle$ を具体化するクラスに依存しないので，$\Delta;\Lambda \not\vdash \texttt{mtype}(m, [\bar{T}/\bar{X}]T)$ となり，仮定と矛盾するため，この Case は起こりえない.

Case OVR-RANGE：

クラス宣言とアスペクト宣言ではメソッド名を排他にしているため，$\texttt{mtype}(m, C\langle\bar{T}\rangle)$ を成立させる $C\langle\bar{T}\rangle$ の M_i に対し，T に $\texttt{mtype}(m, [\bar{T}/\bar{X}]T)$ を成立させる M_j が存在.

$\texttt{mInfo}(M_i, C\langle\bar{T}\rangle) = (\Delta_i, \_, \_, \_, \langle\Lambda_i, m_i, S_i\rangle)$　$\Delta; \mathcal{M} \vdash \texttt{subsump}(C\langle\bar{T}\rangle, \langle\Lambda_i, m_i, S_i\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S_i\rangle)$

$\texttt{mInfo}(M_j, T) = (\Delta_j, \_, \_, \_, \langle\Lambda_j, m_j, S_j\rangle)$　$\Delta; \mathcal{M}_j \vdash \texttt{subsump}(T, \langle\Lambda_j, m_j, S_j\rangle, T, \langle\Lambda, m, S_j\rangle)$

また，OVR-RANGE の定義より，この M_j と，M_i の間で `disconf` 成立.

$\Delta_i, \Delta_j \vdash \texttt{disconf}(T, \langle\Lambda_j, m_j, S_j\rangle, C\langle\bar{X}\rangle, \langle\Lambda_i, m_i, S_i\rangle)$

`disconf` がその各規則で成立する場合ごとに証明.

Case DSC-CLS：$\Delta \vdash C\langle\bar{T}\rangle <: [\bar{T}/\bar{X}]T$ であるため，この Case は起こりえない.

Case DSC-NAME：$\Delta \vdash \texttt{disjoint}(\langle\Lambda_j, m_j, S_j\rangle, \langle\Lambda_i, m_i, S_i\rangle)$

Lemma 1.1 証明時の，disjoint の議論と同様に，`subsump` 成立と `disjoint` 成立が矛盾するため，この Case は起こりえない.

Case DSC-RANGE：$\Delta;\Lambda_j \vdash \texttt{mitype}(m_j, [\bar{T}/\bar{X}]T) = \bar{W} \to W_0$　$\Delta;\Lambda_i \vdash \texttt{mitype}(m_i, C\langle\bar{T}\rangle) = \bar{V} \to V_0$

$\Delta; \mathcal{M} \vdash \texttt{munify}(\langle\Lambda_j, m_j, S_j\rangle, \langle\Lambda_i, m_i, S_i\rangle)$　$\mathcal{M} \vdash W_0{:}\bar{W} = V_0{:}\bar{V}$

Lemma 2 の証明部より，仮定の `subsump` 成立で得られる $\mathcal{M}_i$，$\mathcal{M}_j$ と，`munify` で得られる $\mathcal{M}$ の間に，$Y \in \texttt{pmVars}(\Lambda_i)$, $\mathcal{M}(Y) = T$ `implies` $M_i(Y) = \texttt{mapT}_{\mathcal{M}_j}(T)$ が（i, j が逆の場合にも）成立するので，$\texttt{mapT}_{\mathcal{M}_i}(W_0{:}\bar{W}) = \texttt{mapT}_{\mathcal{M}_j}(V_0{:}\bar{V})$ が成立.

Case IT：クラスとアスペクトでメソッド名を排他にしているため，$\texttt{itype}(m, C\langle\bar{T}\rangle)$ を成立させる M_i^a と，$\texttt{itype}(m, [\bar{T}/\bar{X}]T)$ を成立させる M_j^a が存在する．次の 2 つの場合があり，それぞれ証明を行う.

Case $M_i^a \neq M_j^a$ の場合：

M_1^a，M_j^a 間での，`disconf` 成立から導出.

$\texttt{aInfo}(M_i^a) = (\Delta_i, \_, \_, \_, T_i, \langle\Lambda_i, m_i, S_i\rangle)$　$\texttt{aInfo}(M_j^a) = (\Delta_j, \_, \_, \_, T_j, \langle\Lambda_j, m_j, S_j\rangle)$

$\Delta_i, \Delta_j \vdash \texttt{disconf}(T_i, \langle\Lambda_i, m_i, S_i\rangle, T_j, \langle\Lambda_j, m_j, S_j\rangle)$

$\Delta; \mathcal{M}_i \vdash \texttt{subsump}(T_i, \langle\Lambda_i, m_i, S_i\rangle, C\langle\bar{T}\rangle, \langle\Lambda, m, S_i\rangle)$

$\Delta; \mathcal{M}_j \vdash \texttt{subsump}(T_j, \langle\Lambda_j, m_j, S_j\rangle, [\bar{T}/\bar{X}]T, \langle\Lambda, m, S_j\rangle)$

`disconf` がその各規則で成立する場合ごとに証明.

Case DSC-CLS：$no\ \mathcal{M}$　$\Delta \vdash \texttt{mapT}_{\mathcal{M}}(T_i) <: \texttt{mapT}_{\mathcal{M}}(T_j)$　or　$\Delta \vdash \texttt{mapT}_{\mathcal{M}}(T_j) <: \texttt{mapT}_{\mathcal{M}}(T_i)$

一方，仮定の `subsump` 関係の成立より，$C\langle\bar{T}\rangle = \texttt{mapT}_{\mathcal{M}_i}(T_i)$　$[\bar{T}/\bar{X}]T = \texttt{mapT}_{\mathcal{M}_j}(T_j)$ なる $\mathcal{M}$ が存在し，矛盾するので，この Case は起こりえない.

Case DSC-NAME/RANGE：

MT-CLS の Case での，DSC-NAME/RANGE と同様に証明可能．

Case $M_i^a{=}M_j^a$ の場合：

$\texttt{aInfo}(M_i^a){=}(\Delta,\_,\_,\_,T_i,\langle\Lambda_i,m_i,S_i\rangle)\quad \Delta;\mathcal{M}_i{\vdash}\texttt{subsump}(T_i,\langle\Lambda_i,m_i,S_i\rangle,C{<}\bar{T}{>},\langle\Lambda,m,S_i\rangle)$
$\Delta;\mathcal{M}_i'{\vdash}\texttt{subsump}(T_i,\langle\Lambda_i,m_i,S_i\rangle,[\bar{T}/\bar{X}]T,\langle\Lambda,m,S_i\rangle)$

$C{<}\bar{T}{>}$, $[\bar{T}/\bar{X}]T$ 両方での subsump 成立から，M_i^a の対象クラスは型変数（仮に X）である．

By T-Add：$\Delta\backslash(X{<:}N){\vdash}V_0,\bar{V}$ ok より，シグネチャが対象クラスとなる型変数 X に依存せず，$\texttt{mapT}_{\mathcal{M}_i}(V_0{:}\bar{V}){=}\texttt{mapT}_{\mathcal{M}_j}(V_0{:}\bar{V}){=}U_0{:}\,\bar{U}$ となり，シグネチャは一致する．

Lemma 10（WEAKENING）

$\Delta,\bar{X}{<:}\bar{N}{\vdash}\bar{N}$ ok かつ $\Delta{\vdash}U$ ok のとき，

1) $\Delta{\vdash}S{<:}T$ ならば，$\Delta,\bar{X}{<:}\bar{N}{\vdash}S{<:}T$

2) $\Delta{\vdash}S$ ok ならば，$\Delta,\bar{X}{<:}\bar{N}{\vdash}S$ ok

3) $\Delta;\Gamma;\Lambda{\vdash}e\in T$ ならば，$\Delta;\Gamma,x{\mapsto}U;\Lambda{\vdash}e\in T$ かつ $\Delta,\bar{X}{<:}\bar{N};\Gamma;\Lambda{\vdash}e\in T$

Lemma 11（TYPE MAPPING PRESERVES SUBTYPING）

$\mathcal{M}{=}\bar{Y}{\mapsto}\bar{W}$ $\Delta,\bar{Y}{<:}\bar{P};\bar{Y}{\vdash}\bar{W}{\prec:}\bar{Y}$ $\Delta{\vdash}\bar{W}$ ok なる環境 $\mathcal{M}$ において，$\Delta,\bar{Y}{<:}\bar{P}{\vdash}S{<:}T$ かつ，$\bar{Y}$ が Δ に現れないならば，$\Delta{\vdash}\texttt{mapT}_{\mathcal{M}}(S){<:}\texttt{mapT}_{\mathcal{M}}(T)$

Proof. $\Delta{\vdash}S{<:}T$ の各規則について帰納的に証明する．

Case S-REFL：自明である．

Case S-TRANS：帰納法の仮定から簡単に求まる．

Case S-VAR：$\Delta'{=}\Delta,\bar{Y}{<:}\bar{P}$ $\Delta'(Y){=}C{<}\bar{T}{>}$ $\Delta'{\vdash}Y{<:}C{<}\bar{T}{>}$

以下の 2 つのケースについてそれぞれ証明する．

Case $Y\notin\texttt{dom}(\mathcal{M})$：

このとき，$\Delta(Y){=}C{<}\bar{T}{>}$ $\Delta{\vdash}Y{<:}C{<}\bar{T}{>}$ である．$\bar{Y}$ が Δ に現れないことから，$C{<}\bar{T}{>}$ に $\bar{Y}$ は現れず，$\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>}){=}C{<}\bar{T}{>}$, $\texttt{mapT}_{\mathcal{M}}(Y){=}Y$．よって，$\Delta{\vdash}\texttt{mapT}_{\mathcal{M}}(Y){<:}\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>})$

Case $Y\in\texttt{dom}(\mathcal{M})$：

$\texttt{dom}(\mathcal{M}){=}\bar{Y}$ $\texttt{mapT}_{\mathcal{M}}(\bar{Y}){=}\bar{U}$ に対して，$\texttt{mapT}_{\mathcal{M}}(Y){=}[\bar{U}/\bar{Y}]Y$ $\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>}){=}[\bar{U}/\bar{Y}]C{<}\bar{T}{>}$

Lemma 7（型置換のサブタイプ保存）より，$\Delta{\vdash}[\bar{U}/\bar{Y}]Y{<:}[\bar{U}/\bar{Y}]C{<}\bar{T}{>}$

よって，$\Delta{\vdash}\texttt{mapT}_{\mathcal{M}}(Y){<:}\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>})$ となる．

Case S-CLS：$\Delta,\bar{Y}{<:}\bar{P}{\vdash}C{<}\bar{T}{>}{<:}[\bar{T}/\bar{X}]T\quad CT(C){=}\texttt{class}\ C{<}\bar{X}{\triangleleft}\ \bar{N}{>}\triangleleft T\{\cdots\}$

$\texttt{mapT}_{\mathcal{M}}(\bar{T}){=}\bar{U}$ とすると，$C{<}\bar{U}{>}{=}\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>})$ $[\bar{U}/\bar{X}]T{=}\texttt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}]T)$ に対し，S-CLS より $\Delta{\vdash}C{<}\bar{U}{>}{<:}[\bar{U}/\bar{X}]T$ が成立．

Lemma 12（TYPE MAPPING PRESERVES TYPING）

$\mathcal{M}{=}\bar{Y}{\mapsto}\bar{W}$ $\Delta,\bar{Y}{<:}\bar{P};\bar{Y}{\vdash}\bar{W}{\prec:}\bar{Y}$ $\Delta{\vdash}\bar{W}$ ok なる環境 $\mathcal{M}$ において，$\Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda\ {\vdash}e{\in}T$ かつ，$\bar{Y}$ が Δ に現れないならば，$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda)\ {\vdash}\texttt{mapex}_{\mathcal{M}}(e){\in}W$ と型付け可能で，このとき $\Delta{\vdash}W{<:}\texttt{mapT}_{\mathcal{M}}(T)$ である．

Proof. $\texttt{mapex}_{\mathcal{M}}(e)$ の各規則について帰納的に証明．

Case MAP-VAR：自明である．

Case MAP-FLD：$\texttt{mapex}_{\mathcal{M}}(e_0.f_i){=}\texttt{mapex}_{\mathcal{M}}(e_0).f_i$

By T-FLD：$\Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda{\vdash}e_0.f_i{\in}V_i\quad \Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda{\vdash}e_0{\in}T_0$

$C{<}\bar{T}{>}{=}\texttt{bound}_{\Delta}(T_0),\bar{Y}{<:}\bar{P}\quad \Delta,\bar{Y}{<:}\bar{P}{\vdash}\texttt{fields}(C{<}\bar{T}{>}){=}\bar{V}\ \bar{f}$

帰納法の仮定より，$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}\texttt{mapex}_{\mathcal{M}}(e_0){\in}W_0\quad \Delta{\vdash}W_0{<:}\texttt{mapT}_{\mathcal{M}}(T_0)$

また，$\texttt{mapT}_{\mathcal{M}}(C{<}\bar{T}{>}){=}C{<}\bar{S}{>}$ where $\texttt{mapT}_{\mathcal{M}}(\bar{T}){=}\bar{S}$ に対し，

By fields：$CT(C){=}\texttt{class}\ C{<}\bar{X}{\triangleleft}\ \bar{N}{>}\triangleleft T\{\cdots\bar{U}\ \bar{f}\}\quad \Delta{\vdash}\texttt{fields}(C{<}\bar{S}{>}){=}\cdots,[\bar{S}/\bar{X}]\bar{U}\ \bar{f}$

このとき，$V_i{=}[\bar{T}/\bar{X}]U_i\quad \texttt{mapT}_{\mathcal{M}}(V_i){=}\texttt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}]U_i){=}[\bar{S}/\bar{X}]U_i$ である．

Lemma 8（サブタイプにおける fields の保存）より，

$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda)\ {\vdash}\texttt{mapex}_{\mathcal{M}}(e_0).f_i{\in}W_i{<:}[\bar{S}/\bar{X}]U_i=\texttt{mapT}_{\mathcal{M}}(V_i)$

Case MAP-INV：$\texttt{mapex}_{\mathcal{M}}(e_0.m(\bar{e})){=}e_0'.m(\bar{e}')$ where $e_0'{=}\texttt{mapex}_{\mathcal{M}}(e_0)\quad \bar{e}'{=}\texttt{mapex}_{\mathcal{M}}(\bar{e})$

By T-INVK：$\Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda{\vdash}e_0.m(\bar{e}){\in}V_0\quad \Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda{\vdash}e_0{\in}S$

$\Delta,\bar{Y}{\mapsto}\bar{P};\Gamma;\Lambda{\vdash}\bar{e}{\in}\bar{S}\ \Delta,\bar{Y}{\mapsto}\bar{P}{\vdash}\texttt{mitype}(m,S){=}\bar{V}{\to}V_0\quad \Delta{\vdash}\bar{S}{<:}\bar{V}$

帰納法の仮定より，$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}e_0'{\in}W\quad \Delta{\vdash}W{<:}\texttt{mapT}_{\mathcal{M}}(S)$

$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}\bar{e}'{\in}\bar{U}\quad \Delta{\vdash}\bar{U}{<:}\texttt{mapT}_{\mathcal{M}}(\bar{S})$

Lemma 13（$\mathcal{M}$ 変換の mitype 保存），9（サブタイプにおける mitype 保存）より，

$\Delta;\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}\texttt{mitype}(m,W){=}\texttt{mapT}_{\mathcal{M}}(\bar{V}){\to}\texttt{mapT}_{\mathcal{M}}(V_0)$

Lemma 11（$\mathcal{M}$ 変換の型付け保存）より，$\Delta{\vdash}\bar{U}{<:}\texttt{mapT}_{\mathcal{M}}(\bar{V})$ となり，

よって $\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}e_0'.m(\bar{e}'){\in}\texttt{mapT}_{\mathcal{M}}(V_0)$

Case MAP-NEW：$\texttt{mapex}_{\mathcal{M}}(\texttt{new}\ C{<}\bar{T}{>}(\bar{e})){=}\texttt{new}\ C{<}\bar{T}'{>}(\bar{e}')$ where $\bar{T}'{=}\texttt{mapT}_{\mathcal{M}}(\bar{T})\quad \bar{e}'{=}\texttt{mapex}_{\mathcal{M}}(\bar{e})$

By T-NEW and fields：$\Delta,\bar{Y}{<:}\bar{P};\Gamma;\Lambda{\vdash}\bar{e}\in\bar{S}\quad CT(C){=}\texttt{class}\ C{<}\bar{X}{\triangleleft}\ \bar{N}{>}\triangleleft T\{\cdots\bar{U}\ \bar{f}\}$

$\Delta,\bar{Y}{<:}\bar{P}{\vdash}\texttt{fields}(C{<}\bar{T}{>}){=}[\bar{T}/\bar{X}]\bar{U}\ \bar{f}\quad \Delta,\bar{Y}{<:}\bar{P}{\vdash}\bar{S}{<:}[\bar{T}/\bar{X}]\bar{U}$

$\Delta{\vdash}\texttt{fields}(C{<}\bar{T}'{>}){=}[\bar{T}'/\bar{X}]\bar{U}\ \bar{f}{=}\texttt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}]\bar{U})\ \bar{f}$

帰納法の仮定より，$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}\bar{e}'{\in}\bar{W}\ \ \Delta{\vdash}\bar{W}{<:}\texttt{mapT}_{\mathcal{M}}(\bar{S})$

Lemma 11（$\mathcal{M}$ 変換の型付け保存）より，$\Delta{\vdash}\bar{W}{<:}\texttt{mapT}_{\mathcal{M}}(\bar{S}){<:}\texttt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}]\bar{U})$

よって，$\Delta;\texttt{mapT}_{\mathcal{M}}(\Gamma);\texttt{mapT}_{\mathcal{M}}(\Lambda){\vdash}\texttt{new}\ C{<}\bar{T}'{>}(\bar{e}'){\in}C{<}\bar{T}'{>}$

Case MAP-CAST：$\mathtt{mapex}_{\mathcal{M}}((T)e_0)=(T')\mathtt{mapex}_{\mathcal{M}}(e_0)$ where $T'=\mathtt{mapT}_{\mathcal{M}}(T)$

By T-CAST：$\Delta, \bar{Y}<:\bar{P}; \Gamma; \Lambda \vdash (T)e_0 \in T$　$\Delta, \bar{Y}<:\bar{P}; \Gamma; \Lambda \vdash e_0 \in T_0$

帰納法の仮定より，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Gamma); \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mapex}_{\mathcal{M}}(e_0) \in W$　$\Delta \vdash W <: \mathtt{mapT}_{\mathcal{M}}(T_0)$

よって，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Gamma); \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash (T')\mathtt{mapex}_{\mathcal{M}}(e) \in T'$

Case MAP-ORG：MAP-INV と同様に証明可能．

Lemma 13

$\mathcal{M}=\bar{Y} \mapsto \bar{W}$　$\Delta, \bar{Y}<:\bar{P}; \bar{Y} \vdash \bar{W} \prec: \bar{Y}$　$\Delta \vdash \bar{W}\,\mathtt{ok}$ なる環境 $\mathcal{M}$ において，

$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mitype}(m, T)=\bar{V} \rightharpoonup V_0$ かつ，$\bar{Y}$ が Δ に現れないならば，

$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mitype}(m, \mathtt{mapT}_{\mathcal{M}}(T))=\mathtt{mapT}_{\mathcal{M}}(\bar{V}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}}(V_0)$

Proof. $\mathtt{mitype}(m, T)$ の各ルールに対し，帰納的に証明．

Case MT-Λ：$\mathcal{M}$ 変換前は，$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mitype}(m, T)=\bar{V} \rightharpoonup V_0$　$\Lambda=T \mapsto \langle \_, \_, V_0\ m(\bar{V}) \rangle$

一方 $\mathcal{M}$ 変換後は，$\mathtt{mapT}_{\mathcal{M}}(T)=W$　$\mathtt{mapT}_{\mathcal{M}}(\Lambda)=W \mapsto \langle \_, \_, \mathtt{mapT}_{\mathcal{M}}(V_0)\ m(\mathtt{mapT}_{\mathcal{M}}(\bar{V})) \rangle$

より，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mitype}(m, W)=\mathtt{mapT}_{\mathcal{M}}(\bar{V}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}}(V_0)$ となる．

Case MT-CLS：

MT-CLS 規則より，以下の条件が成立する．

$\mathtt{mInfo}(M_i, C<\bar{X} \triangleleft \bar{N}>)=(\Delta', \_, \_, \bar{S} \rightharpoonup S_0, \langle \Lambda_i, m_i, S'_i \rangle)$

$\Delta_k=\Delta, \bar{Y}<:\bar{P}, [\bar{T}/\bar{X}](\Delta')$　$\Lambda_k=[\bar{T}/\bar{X}](\Lambda_i)$　$S_k=[\bar{T}/\bar{X}]S'_i$

$\Delta_k; \mathcal{M}_i \vdash \mathtt{subsump}(C<\bar{T}>, \langle \Lambda_k, m_i, S_k \rangle, C<\bar{T}>, \langle \Lambda, m, S_k \rangle)$

$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mtype}(m, C<\bar{T}>)=[\bar{T}/\bar{X}](\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0))$

subsump が，sump-S/R で成立する場合に分け，

$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mtype}(m, C<\bar{T}'>)=\mathtt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}](\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)))$

where $C<\bar{T}'>=\mathtt{mapT}_{\mathcal{M}}(C<\bar{T}>)$　$\bar{T}'=\mathtt{mapT}_{\mathcal{M}}(\bar{T})$ を証明．

Case sump-S：$\mathcal{M}_i=\phi$ で成立するため，

$\mathcal{M}$ 変換前は，$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mtype}(m, C<\bar{T}>)=[\bar{T}/\bar{X}](\bar{S} \rightharpoonup S_0)$

一方，$\mathcal{M}$ 変換後は，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mtype}(m, C<\bar{T}'>)=[\bar{T}'/\bar{X}](\bar{S} \rightharpoonup S_0)$

このとき，$[\bar{T}'/\bar{X}](\bar{S} \rightharpoonup S_0)=\mathtt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}](\bar{S} \rightharpoonup S_0))$

Case sump-R：$\mathtt{dropPre}(\langle \Lambda_k, m_i \rangle, \langle \Lambda, m \rangle)=m'_i : m'$　$\bar{Y}'=\mathtt{pmVars}(\Lambda_k)$　$\Delta_k; \Lambda_k \vdash \mathtt{mtype}(m'_i, S_k)=\bar{U} \rightharpoonup U_0$

$\Delta_k; \Lambda \vdash \mathtt{mtype}(m', S_k)=\bar{V}' \rightharpoonup V'_0$　$\Delta_k; \mathcal{M}_i; \bar{Y}' \vdash \mathtt{tunify}(C<\bar{T}>:U_0:\bar{U}, C<\bar{T}>:V'_0:\bar{V}')$

に対し，$\mathcal{M}$ 変換前は，$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mtype}(m, C<\bar{T}>)=[\bar{T}/\bar{X}](\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0))$．

一方，帰納法の仮定より，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mtype}(m', \mathtt{mapT}_{\mathcal{M}}(S_k))=\mathtt{mapT}_{\mathcal{M}}(\bar{V}') \rightharpoonup \mathtt{mapT}_{\mathcal{M}}(V'_0)$ であり，

その際，$\Delta_k; \mathcal{M}_j; \bar{Y}' \vdash \mathtt{tunify}(C<\bar{T}>:U_0:\bar{U}, C<\bar{T}>:\mathtt{mapT}_{\mathcal{M}}(V'_0):\mathtt{mapT}_{\mathcal{M}}(\bar{V}'))$ を成立させる $\mathcal{M}_j$ と，$\mathcal{M}_i$ の間には，$\mathcal{M}_i=\bar{Y}' \mapsto \bar{W}'$ ならば $\mathcal{M}_j=\bar{Y}' \mapsto \mathtt{mapT}_{\mathcal{M}}(\bar{W}')$ という関係がある．

よって，$\mathcal{M}$ 変換後は，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mitype}(m, C<\bar{T}'>)=$

$$[\bar{T}'/\bar{X}](\mathtt{mapT}_{\mathcal{M}_j}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_j}(S_0)) = \mathtt{mapT}_{\mathcal{M}}([\bar{T}/\bar{X}](\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)))$$

Case IT：

IT 規則より，以下の条件が成立する．

$\mathtt{aInfo}(M^a_i)=(\Delta', \_, \_, \bar{S} \rightharpoonup S_0, T', \langle \Lambda', m', S' \rangle)$　$\Delta', \Delta, \bar{Y}<:\bar{P}; \mathcal{M} \vdash \mathtt{subsump}(T', \langle \Lambda', m', S' \rangle, T, \langle \Lambda, m, S' \rangle)$

$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{itype}(m, T)=\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)$

subsump が，sump-S/R で成立する場合に分け，

$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{itype}(m, W)=\mathtt{mapT}_{\mathcal{M}}((\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)))$ where $W=\mathtt{mapT}_{\mathcal{M}}(T)$ を証明．

Case sump-S：$\Delta', \Delta, \bar{Y}<:\bar{P}; \mathcal{M}_i; \bar{Z} \vdash \mathtt{tunify}(T', T)$

に対し，$\mathcal{M}$ 変換前は，$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{itype}(m, T)=\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)$

一方 $\mathcal{M}_i=\bar{Z} \mapsto \bar{T}$ に対し，$\Delta', \Delta; \mathcal{M}_j; \bar{Z} \vdash \mathtt{tunify}(T', W)$ for $\mathcal{M}_j=\bar{Z} \mapsto \mathtt{mapT}_{\mathcal{M}}(\bar{T})$ が成立する．$\mathcal{M}$ 変換後は，$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mitype}(m, W)=\mathtt{mapT}_{\mathcal{M}_j}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_j}(S_0)=\mathtt{mapT}_{\mathcal{M}}(\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0))$

Case sump-R：$\Delta', \Delta, \bar{Y}<:\bar{P}; \mathcal{M}'_i; \bar{Z} \vdash \mathtt{tunify}(T', T)$　$\mathtt{dropPre}(\langle \Lambda_i, m_i \rangle, \langle \Lambda, m \rangle)=m'_i : m'$

$S'_i=\mathtt{mapT}_{\mathcal{M}'_i}(S')$　$\Delta', \Delta, \bar{Y}<:\bar{P}; \Lambda_i \vdash \mathtt{mtype}(m'_i, S')=\bar{U} \rightharpoonup U_0$　$\Delta', \Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{mtype}(m', S'_i)=\bar{V}' \rightharpoonup V'_0$

$\bar{Y}'=\mathtt{pmVars}(\Lambda_i)$　$\Delta', \Delta, \bar{Y}<:\bar{P}; \mathcal{M}_i; \bar{Y}' \vdash \mathtt{tunify}(T' : U_0 : \bar{U}, T : V'_0 : \bar{V}')$

に対し，$\mathcal{M}$ 変換前は，$\Delta, \bar{Y}<:\bar{P}; \Lambda \vdash \mathtt{itype}(m, T)=\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0)$

一方，$\mathcal{M}$ 変換後については，$\Delta', \Delta; \mathcal{M}'_j; \bar{Z} \vdash \mathtt{tunify}(T', W)$　$S'_j=\mathtt{mapT}_{\mathcal{M}'_j}(S')=\mathtt{mapT}_{\mathcal{M}}(S'_i)$ および，帰納法の仮定より，$\Delta', \Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mtype}(m', S'_j)=\mathtt{mapT}_{\mathcal{M}}(\bar{V}' \rightharpoonup V'_0)$ である．

その際，$\Delta', \Delta; \mathcal{M}_j; \bar{Y}' \vdash \mathtt{tunify}(T' : U_0 : \bar{U}, T : \mathtt{mapT}_{\mathcal{M}}(V'_0) : \mathtt{mapT}_{\mathcal{M}}(\bar{V}'))$ を成立させる $\mathcal{M}_j$ と，$\mathcal{M}_i$ の間には，$\mathcal{M}_i=\bar{Y}' \mapsto \bar{T}$ ならば $\mathcal{M}_j=\bar{Y}' \mapsto \mathtt{mapT}_{\mathcal{M}}(\bar{T})$ という関係がある．

よって，$\mathcal{M}$ 変換後は，

$\Delta; \mathtt{mapT}_{\mathcal{M}}(\Lambda) \vdash \mathtt{mitype}(m, W) = \mathtt{mapT}_{\mathcal{M}_j}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_j}(S_0) = \mathtt{mapT}_{\mathcal{M}}(\mathtt{mapT}_{\mathcal{M}_i}(\bar{S}) \rightharpoonup \mathtt{mapT}_{\mathcal{M}_i}(S_0))$

Case MT-SUP/IT-SUP：帰納的に，簡単に求まる．

Lemma 14（NAME SUBSTITUTION PRESERVES METHOD LOOKUP）

$\Lambda'=S \mapsto \langle \_, \_, W_0\ u(\bar{W}) \rangle$ かつ $\Delta; \Lambda \vdash \mathtt{mitype}(m, S)=\bar{W} \rightharpoonup W_0$ のとき，

$\Delta; \Lambda' \vdash \mathtt{mitype}(m_0, T_0)=\bar{V} \rightharpoonup V_0$ が成立するならば，$\Delta; \Lambda \vdash \mathtt{mitype}([m/u]m_0, T_0)=\bar{V} \rightharpoonup V_0$

Proof. $\Delta; \Lambda' \vdash \mathtt{mitype}(m_0, T_0)$ が，mitype の各規則で成立した場合ごとに，帰納的に証明．

Case MT-Λ：このケースは，$m_0=u$　$T_0=S$ のときで，

名前置換前の $\Delta; \Lambda' \vdash \mathtt{mtype}(m_0, T_0)=\mathtt{mtype}(u, S)=\bar{W} \rightharpoonup W_0$ に対し，

名前置換後の $\Delta;\Lambda\vdash\texttt{mtype}([m/u]m_0,T_0)=\texttt{mtype}(m,S)=\bar{W}\to W_0$ は明らかである．

Case　MT-CLS：$\Delta;\Lambda\vdash\texttt{mtype}(m_0,C\!<\!\bar{T}\!>)=[\bar{T}/\bar{X}](\texttt{mapT}_{\mathcal{M}_i}(\bar{S})\to\texttt{mapT}_{\mathcal{M}_i}(S_0))$

$\texttt{mInfo}(M_i,C\!<\!\bar{X}\lhd\bar{N}\!>)=(\Delta',\_,\_,\bar{S}\to S_0,\langle\Lambda_i,m_i,S_i\rangle)$

$\Delta_k=\Delta,[\bar{T}/\bar{X}](\Delta')\quad\Lambda_k=[\bar{T}/\bar{X}](\Lambda_i)\quad S_k=[\bar{T}/\bar{X}]S_i$

$\Delta_k;\mathcal{M}_i\vdash\texttt{subsump}(C\!<\!\bar{T}\!>,\langle\Lambda_k,m_i,S_k\rangle,C\!<\!\bar{T}\!>,\langle\Lambda',m_0,S_k\rangle)$

subsumpが，sump-S/Rで成立する場合ごとに証明する．

Case　sump-S：$m_i=m_0$ $\Lambda\vdash\texttt{static}(m_0)$ なので，m_0 に名前変数 u は含まれず，$m_0=[m/u]m_0$ である．sump-S では，Λ によらず $\mathcal{M}_i=\phi$ で成立するので，名前置換の前後でシグネチャは変わらず，

$\Delta;\Lambda'\vdash\texttt{mtype}(m_0,C\!<\!\bar{T}\!>)=[\bar{T}/\bar{X}](\bar{S}\to S_0)\quad\Delta;\Lambda\vdash\texttt{mtype}([m/u]m_0,C\!<\!\bar{T}\!>)=[\bar{T}/\bar{X}](\bar{S}\to S_0)$

Case　sump-R：$\Delta;\mathcal{M}_i;\bar{Y}\vdash\texttt{tunify}(T{:}U_0{:}\bar{U},T_0{:}V_0'{:}\bar{V}')\quad\texttt{dropPre}(\langle\Lambda_i,m_i\rangle,\langle\_,m_0\rangle)=m_i'{:}m_0'$

$\Delta_k;\Lambda_k\vdash\texttt{mtype}(m_i',S_k)=\bar{U}\to U_0\quad\Delta_k;\Lambda'\vdash\texttt{mtype}(m_0',S_k)=\bar{V}'\to V_0'$ が名前置換前に成立．

帰納法の仮定より，$\Delta_k;\Lambda\vdash\texttt{mtype}([m/u]m_0',S_k)=\bar{V}'\to V_0'$ であり，名前置換後も，同じ $\mathcal{M}_i$ で sump-R が成立するので，$\Delta;\Lambda\vdash\texttt{mtype}([m/u]m_0,C\!<\!\bar{T}\!>)=[\bar{T}/\bar{X}](\texttt{mapT}_{\mathcal{M}_i}(\bar{S})\to\texttt{mapT}_{\mathcal{M}_i}(S_0))$

Case　IT：$\Delta;\Lambda\vdash\texttt{itype}(m_0,T_0)=\texttt{mapT}_{\mathcal{M}_i}(\bar{S})\to\texttt{mapT}_{\mathcal{M}_i}(S_0)$

$\texttt{aInfo}(M_i^a)=(\Delta',\_,\_,\bar{S}\to S_0,T_i,\langle\Lambda_i,m_i,S'\rangle)\quad\Delta',\Delta;\mathcal{M}_i\vdash\texttt{subsump}(T_i,\langle\Lambda_i,m_i,S'\rangle,T_0,\langle\Lambda',m,S'\rangle)$

subsump が，sump-S/R で成立する場合ごとに証明する．

Case　sump-S：MT-CLS のときと同様に，$m_0=[m/u]m_0$ である．このとき，名前置換の前後で，同一の $\mathcal{M}_i$ で sump-S が成立するので，

$\Delta;\Lambda\vdash\texttt{itype}([m/u]m_0,T_0)=\texttt{mapT}_{\mathcal{M}_i}(\bar{S})\to\texttt{mapT}_{\mathcal{M}_i}(S_0)$

Case　sump-R：$\Delta',\Delta;\mathcal{M}_i;\bar{Y}\vdash\texttt{tunify}(T{:}U_0{:}\bar{U},T_0{:}V_0'{:}\bar{V}')\quad\texttt{dropPre}(\langle\Lambda_i,m_i\rangle,\langle\_,m_0\rangle)=m_i'{:}m_0'$

$\Delta',\Delta;\Lambda_i\vdash\texttt{mtype}(m_i',S')=\bar{U}\to U_0\quad\Delta',\Delta;\Lambda'\vdash\texttt{mtype}(m_0',S_i')=\bar{V}'\to V_0'$ が名前置換前に成立．

帰納法の仮定より，$\Delta;\Lambda\vdash\texttt{mtype}([m/u]m_0',S_i')=\bar{V}'\to V_0'$ であり，名前置換後も，同じ $\mathcal{M}_i$ で sump-R が成立するので，$\Delta;\Lambda\vdash\texttt{mtype}([m/u]m_0,T_0)=\texttt{mapT}_{\mathcal{M}_i}(\bar{S})\to\texttt{mapT}_{\mathcal{M}_i}(S_0)$

Case　MT-SUP/IT-SUP：帰納的に，簡単に求まる．

Lemma 15（TYPE SUBSTITUTION PRESERVES TYPING）

$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e\in T$, $\Delta_1\vdash\bar{U}<:[\bar{U}/\bar{X}]\bar{N}$，かつ $\Delta_1\vdash\bar{U}$ **ok** かつ $\bar{X}$ が Δ_1 に現れないならば，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]e\in S$ と型付け可能で，このとき $\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash S<:[\bar{U}/\bar{X}]T$

Proof.　e の各型付け規則について帰納的に証明する．

Case　T-VAR：$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash x\in T\quad\Gamma(x)=T$

$[\bar{U}/\bar{X}]\Gamma(x)=[\bar{U}/\bar{X}]T$ なので，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash x\in[\bar{U}/\bar{X}]T$

Case　T-FLD：$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e_0.f_i\in V_i$ where $\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e_0\in T_0$

$C\!<\!\bar{T}\!>=\texttt{bound}_{(\Delta_1,\bar{X}<:\bar{N},\Delta_2)}(T_0)\quad\Delta_1,\bar{X}<:\bar{N},\Delta_2\vdash\texttt{fields}(C\!<\!\bar{T}\!>)=\cdots\bar{V}\ \bar{f}$

帰納法の仮定より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]e_0\in[\bar{U}/\bar{X}]T_0$

Lemma 7（型置換のサブタイプ保存）より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash[\bar{U}/\bar{X}]T_0<:[\bar{U}/\bar{X}]C\!<\!\bar{T}\!>$

このとき，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash\texttt{fields}([\bar{U}/\bar{X}]C\!<\!\bar{T}\!>)=\cdots[\bar{U}/\bar{X}]\bar{V}\ \bar{f}$ なので，

$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]e_0.f\in S_i\quad\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash S_i<:[\bar{U}/\bar{X}]V_i$

Case　T-INVK：$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e_0.m(\bar{e})\in V_0$

where $\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e_0\in T_0\quad\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Lambda\vdash\texttt{mitype}(m,T_0)=\bar{V}\to V_0$

$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash\bar{e}\in\bar{T}\quad\Delta_1,\bar{X}<:\bar{N},\Delta_2\vdash\bar{T}<:\bar{V}$

帰納法の仮定より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]\bar{e}\in\bar{W}<:[\bar{U}/\bar{X}]\bar{V}$

$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]e_0\in W_0<:[\bar{U}/\bar{X}]T_0$

また，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;\Lambda\vdash\texttt{mitype}(m,[\bar{U}/\bar{X}]T_0)=[\bar{U}/\bar{X}](\bar{V}\to V_0)$ であることが，Lemma 13 と同様にして証明可能である（詳細略）．よって，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}](e_0.m(\bar{e}))\in[\bar{U}/\bar{X}]V_0$

Case　T-NEW：$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash\texttt{new}\,C\!<\!\bar{T}\!>(\bar{e})\in C\!<\!\bar{T}\!>$

where $\Delta_1,\bar{X}<:\bar{N},\Delta_2\vdash\texttt{fields}(C\!<\!\bar{T}\!>)=\bar{V}\ \bar{f}\quad\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash\bar{e}\in\bar{S}<:\bar{V}$

帰納法の仮定より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}]\bar{e}\in\bar{W}\quad\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash\bar{W}<:[\bar{U}/\bar{X}]\bar{S}$

By **fields**：$\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash\texttt{fields}([\bar{U}/\bar{X}]C\!<\!\bar{T}\!>)=[\bar{U}/\bar{X}]\bar{V}\ \bar{f}$

Lemma 7（型置換のサブタイプ保存）より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash\bar{W}<:[\bar{U}/\bar{X}]\bar{S}<:[\bar{U}/\bar{X}]\bar{V}$

よって，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash[\bar{U}/\bar{X}](\texttt{new}\,C\!<\!\bar{T}\!>(\bar{e}))\in[\bar{U}/\bar{X}]C\!<\!\bar{T}\!>$

Case　T-CAST：$\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash(T)e_0\in T$ where $\Delta_1,\bar{X}<:\bar{N},\Delta_2;\Gamma;\Lambda\vdash e_0\in T_0$

帰納法の仮定より，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash e_0'\in W_0\quad\Delta_1,[\bar{U}/\bar{X}]\Delta_2\vdash W_0<:[\bar{U}/\bar{X}]T_0$

よって，$\Delta_1,[\bar{U}/\bar{X}]\Delta_2;[\bar{U}/\bar{X}]\Gamma;[\bar{U}/\bar{X}]\Lambda\vdash([\bar{U}/\bar{X}]T)[\bar{U}/\bar{X}]e_0\in[\bar{U}/\bar{X}]T$

Case　T-ORG：T-INVK と同様に証明可能．

(平成 20 年 9 月 29 日受付)
(平成 20 年 12 月 26 日採録)

草野　直樹

1984 年生．2007 年神戸大学工学部情報知能工学科卒業．現在，同大学大学院工学研究科情報知能学専攻在学中．プログラミング言語処理系等に興味を持つ．

鎌田十三郎（正会員）

1970 年生．1993 年東京大学理学部情報科学科卒業．1995 年同大学大学院理学系研究科情報科学専攻修士課程修了．1998 年同博士課程単位修得退学．1996〜1998 年日本学術振興会特別研究員（東京大学）．1998 年より神戸大学工学部助手．博士（工学）．並列・分散処理，言語処理系等に興味を持つ．日本ソフトウェア科学会，ACM 各会員．