Featherweight Java のための漸進的型付け

伊奈 林太郎 [1]　　五十嵐 淳 [2]

[1] 京都大学 工学部情報学科
ina@kuis.kyoto-u.ac.jp
[2] 京都大学 大学院情報学研究科
igarashi@kuis.kyoto-u.ac.jp

概 要 静的型システムと動的型システムの両者の利点を活かす枠組みとして, Siek と Taha は漸進的型付けを提唱している. 漸進的型付けでは, 型宣言された部分のみ静的型検査が行なわれ, 残りの部分については実行時検査が行なわれる. これにより, 当初型を付けずに書いたプログラムに型宣言を徐々に付加し, 静的型付けされたプログラムを完成させることができる. 本研究では, 漸進的型付けをクラスに基づくオブジェクト指向言語で実現する理論的基盤として, Igarashi, Pierce, Wadler らの計算体系 Featherweight Java (FJ) に動的型を導入した体系 $FJ^?$ を定義し, 型付け規則を与える. さらに $FJ^?$ から FJ にリフレクションを加えた体系への変換を定義することで意味論を与え, 静的に検査した部分の安全性が保証されることを示す.

1 はじめに

1.1 背景

静的型システムと動的型システムにはそれぞれの長所と短所がある. 前者はプログラマの間違いをコンパイル時に検出するが, 型の整合性を意識した開発を要求する. 後者は型を意識しない素早い開発を可能にするが, 実行時検査のためにパフォーマンスが低下する. このような静的型システムと動的型システム両者の利点を活かすための取り組みは, これまでにも多数行なわれている. その中の一つとして, Siek と Taha [1] によって提案された両者の中間の枠組みが漸進的型付け (Gradual Typing) である. 漸進的型付けでは, プログラマは静的型検査を行なうべき部分とそうでない部分を明示的に指定でき, その指定に従って, 処理系は型宣言された部分のみ静的型検査を行ない, 残りの部分については実行時検査を行なう. これにより, 元々は型を付けずに書いたプログラムに型宣言を徐々に付加し, 最終的に完全に型付けされたプログラムを完成させるというスタイルの開発が可能になる. さらに, 部分的に型宣言された状態でも, 型宣言された部分については実行の過程で型エラーを生じないことが保証できる.

1.2 本研究の目的

本研究では, 漸進的型付けを Java のようなクラスに基づくオブジェクト指向言語で実現する理論的基盤として, Igarashi, Pierce, Wadler [2] らの Java を単純化した計算体系 Featherweight Java (FJ) に動的型を導入した体系 $FJ^?$ を定義する. $FJ^?$ では静的型検査を行なう部分と行なわない部分を明示するために, 静的型検査の対象としない部分を動的型?として宣言できるようにする. $FJ^?$ に意味論を与えたとき, 部分的に動的型を使うことを許しつつ, 静的に検査した部分に関しては実行時の型エラーを生じないことを示すのが本研究の目的である. そのために, 実行時検査をキャストとリフレクションによって明示した体系 FJ_{refl} を定義し, この体系では項が正規形に簡約されたときにキャストおよびリフレクションが失敗していなければ必ず値が得られ, しかもその値の型が簡約前の項の型と無関係なものにはならないという性質を証明する. そして $FJ^?$ から FJ_{refl} への変換を定義することで, この性質を満たす意味論を $FJ^?$ に与える.

1.3 FJ$^?$ の概要

静的型検査 FJ$^?$ では, 静的型検査の対象とならない部分を指定するために, 実行時に検査される動的型を表す?型を変数や返り値の型宣言時にクラス名の代わりに用いる. ?型はワイルドカードのような役割を果たし, どんな型の式が要求される場面でも?型の式を使用でき, ?型の式を要求する場面ではどんな型の式も使うことができる. ?型の式が指すオブジェクトが実際に要求されている型に対して妥当なものかどうかは実行時に検査される. また, ?型が与えられた式に対しては, どんなフィールドアクセス, メソッド呼出しも静的型検査でエラーになることはなく, その式が指すオブジェクトが該当するフィールドやメソッドを持つかどうかは実行時に検査される.

FJ$^?$ での漸進的型付けの概念を説明するために, 簡単な例を示す.

```
class X extends Object {
  X() { super(); }
  Object m(A x) {
    return x.f;
  }
}
class Y extends Object {
  Y() { super(); }
  Object m(? x) {
    return x.f;
  }
}
```

```
class A extends Object {
  Object f;
  A(Object f) { super(); this.f=f; }
}
class B extends Object {
  Object f;
  B(Object f) { super(); this.f=f; }
}
class C extends Object {
  C() { super(); }
}
```

クラス X は, クラス A のオブジェクトを受け取ってそのフィールド f を返すようなメソッド m を持つクラスである. クラス Y は, クラス X でクラス A を指定している部分を動的型?にしたクラスで, なんらかのオブジェクトを受け取ってそのフィールド f を返すようなメソッド m を持つ. クラス A, B はフィールド f を持つクラスで, クラス C はフィールドもメソッドも持たないクラスである.

このとき,

```
new X().m(new A(new Object()));  // OK
new X().m(new B(new Object()));  // reject
```

のように, クラス X のメソッド m の引数としてクラス A のオブジェクトを与えることはできるが, クラス B のオブジェクトを与えた場合は静的型検査の時点で受理されない. 一方,

```
new Y().m(new A(new Object()));  // OK
new Y().m(new B(new Object()));  // OK
```

のような場合は, Y.m の引数には動的型が指定されているので, クラス B のオブジェクトも受理される. この場合はクラス B は実際にフィールド f を持つので実行も正しく行なわれる. また,

```
new Y().m(new C());  // accept
```

のように, フィールド f を持たないオブジェクトを与えた場合も, 型検査器は new C() を動的型と判断するので, 静的型検査では受理される. (もちろん実行時にエラーになる.) ただし, 動的型が期待される場所にどんな式を書いてもよいわけではなく,

```
new Y().m(new C().foo);  // reject
```

のような式は, new C().foo 自体が (C はフィールド foo を持たず) 静的型検査を通らないため, 受理されない.

変換によるプログラムの実行 型検査に通った $\mathrm{FJ}^?$ 項の評価は, FJ にリフレクションを加えた $\mathrm{FJ}_{\mathrm{refl}}$ の項に変換し, $\mathrm{FJ}_{\mathrm{refl}}$ の項を評価することで行なわれる. このとき, 動的型?の関与する項によって発生するエラーは, 動的に型の付いた部分と静的に型の付いた部分の境界で検出される. このことについて, 簡単な例を用いて説明する.

前述の例で用いたクラスに加えて次のクラス W を定義する.

```
class W extends Object {
  ? f;
  W(? f) { super(); this.f=f; }
}
```

このとき, new W(new C()).f には動的型?が付くので, new X().m(new W(new C()).f) は静的型検査で受理される. この式を評価する場合, 次のようにキャストの挿入された式に変換される.

$$\texttt{new X().m(new W(new C()).f)} \rightsquigarrow \texttt{new X().m((A)new W(new C()).f)}$$

メソッド m のレシーバの型は X なので, クラス定義からメソッド X.m の型が A → Object であることが分かり, キャスト先を A にすべきことが分かる. 変換された式は

$$\texttt{new X().m((A)new W(new C()).f)} \longrightarrow_{\mathrm{R}} \texttt{new X().m(}\boxed{\texttt{(A)new C()}}\texttt{)}$$

と簡約されるが, (A)new C() は型の合わない不正なキャストになるので, 動的型の式を A 型の引数に渡している箇所でエラーが検出されることになる.

また, new W(new X()).f は動的型なので, (new W(new X()).f).m(new C()) という式には型が付く. この式を評価する場合, 次のようにリフレクションを用いた式に変換される.

$$\texttt{new W(new X()).f.m(new C())} \rightsquigarrow \mathit{invoke}\texttt{(new W(new X()).f, m, new C())}$$

invoke(e, m, e′) がリフレクションによるメソッド呼び出しを表しており, オブジェクト e のメソッド m を引数 e′ で呼び出す, という意味である. この式は

$$\begin{aligned}\mathit{invoke}\texttt{(new W(new X()).f, m, new C())} &\longrightarrow_{\mathrm{R}} \mathit{invoke}\texttt{(new X(), m, new C())}\\ &\longrightarrow_{\mathrm{R}} [\texttt{((A)new C())/x}]\texttt{x.f}\\ &= \quad \texttt{(}\boxed{\texttt{(A)new C()}}\texttt{).f}\end{aligned}$$

と簡約される. リフレクションによる呼び出しが簡約されるときには, レシーバにそのメソッドがあることが検査され, さらに, 実引数の型が正しいことを実行時に検査するためのキャストが挿入される. それによって (A)new C() が型の合わない不正なキャストになるので, ここでエラーが検出される.

1.4 本稿の構成

まず第 2 節では, $\mathrm{FJ}^?$ と $\mathrm{FJ}_{\mathrm{refl}}$ のもとになる FJ の定義と性質について説明する. 第 3 節では, $\mathrm{FJ}^?$ の構文と型付け規則を定義し, FJ に含まれる項の型判断は $\mathrm{FJ}^?$ においても全く同様になることを示す. 第 4 節では, 変換先である $\mathrm{FJ}_{\mathrm{refl}}$ の構文, 型付け規則, 簡約規則を定義してから, 変換を定義する. それから, 変換の性質, $\mathrm{FJ}_{\mathrm{refl}}$ の型安全性の性質を証明し, 最後に型付けされた $\mathrm{FJ}^?$ の項を型付けされた $\mathrm{FJ}_{\mathrm{refl}}$ の項へ変換できることを示す. 第 5 節で関連研究について議論し, 第 6 節で結論と今後の課題を述べる.

2 FJ

FJ は, 型に関する厳密な議論を行なうために Java を単純化した体系で, Igarashi, Pierce, Wadler [2] によって提案された. 本節では, まず FJ を定義し, その基本的性質を紹介する.

2.1 構文

FJ のプログラムは, クラステーブル CT と, 項 e の組 $(CT, \mathtt{e})$ からなる. CT は, クラス名 C から同名のクラスの定義 L への写像で, Object クラスだけは CT の定義域には含まれず, CT に現れる Object 以外のクラス名は CT の定義域に入っている. また CT 内の継承関係は循環しない. 構文の定義では 0 個以上の要素の列を略記して表す. $\overline{\mathtt{f}}$ と書けば $\mathtt{f}_1\mathtt{,f}_2\mathtt{,...,f}_n$ を表す ($\overline{\mathtt{C}}$, $\overline{\mathtt{x}}$, $\overline{\mathtt{e}}$ などについても同様). ただし $\overline{\mathtt{M}}$ はコンマなしの $\mathtt{M}_1...\mathtt{M}_n$ を表す. 列 $\overline{\mathtt{f}}$ の長さ n は $\#(\overline{\mathtt{f}})$ と表す. 組の列 "$\mathtt{C}_1\ \mathtt{f}_1\mathtt{,...,C}_n\mathtt{f}_n$" は "$\overline{\mathtt{C}}\ \overline{\mathtt{f}}$", "$\mathtt{C}_1\ \mathtt{f}_1\mathtt{;...C}_n\mathtt{f}_n\mathtt{;}$" は "$\overline{\mathtt{C}}\ \overline{\mathtt{f}}\mathtt{;}$", と略記する. 空列は • で表す. また, 列中のフィールド名, メソッド名の重複はないものとする. あるクラスで定義されたフィールドを, それを継承するサブクラスで再定義することはできない. 同名のメソッドはオーバーライドと見なされる.

定義 1 (クラス定義と項). クラス定義 L と項 e は次のように定義される.

```
T   ::=  C
L   ::=  class C extends C {T̄ f̄; K M̄}
K   ::=  C(T̄ f̄){super(f̄); this.f̄=f̄;}
M   ::=  T m(T̄ x̄){ return e; }
e   ::=  x  |  e.f  |  e.m(ē)  |  new C(ē)  |  (C)e
```

クラス定義 L 中の $\overline{\mathtt{T}}\ \overline{\mathtt{f}}$ がフィールド, K がコンストラクタ, $\overline{\mathtt{M}}$ がメソッドである. メソッド定義 $\overline{\mathtt{M}}$ は m がメソッド名, T が返り値のクラス, $\overline{\mathtt{T}}\ \overline{\mathtt{x}}$ が引数の型と変数名, e がメソッドの本体の項を表している. 項 e は変数 x, フィールドアクセス e.f, メソッド呼出し $\mathtt{e.m}(\overline{\mathtt{e}})$, オブジェクトコンストラクタ $\mathtt{new\ C}(\overline{\mathtt{e}})$, キャスト $\mathtt{(C)e}$ のうちのいずれかである. FJ では型として使用できるのはクラス名のみであるが, 次節以降の拡張を考慮にいれて型の構文規則を用意してある.

ここで次節以降で使用するメタ変数の意味を整理しておく.

範囲	型名	クラス名	フィールド名	メソッド名	変数	項
メタ変数	S, T, U, V	C, D	f, g	m	x	e, d

以下の議論では CT は暗黙に仮定されており, 規則中の class C extends D {...}は記述を簡単にするためのもので, $CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{...\}}$ のことである.

2.2 型付け規則

FJ において, すべてのクラスはちょうど 1 つのスーパクラスを持つが, `Object` クラスだけはクラステーブル外で定義されており, スーパクラスを持たない. クラスの部分型関係 $<:$ は継承の宣言 (`extends` 節) から導かれる反射的推移的閉包である.

定義 2 (部分型 (subtyping)). FJ の部分型関係 $<:$ は次のように定義される.

$$\mathtt{C} <: \mathtt{C} \qquad \frac{\mathtt{C} <: \mathtt{D} \quad \mathtt{D} <: \mathtt{E}}{\mathtt{C} <: \mathtt{E}} \qquad \frac{\mathtt{class\ C\ extends\ D\ \{...\}}}{\mathtt{C} <: \mathtt{D}}$$

次に, 型付け規則と簡約規則で CT を参照する際の補助関数を定義する.

定義 3 (補助関数).

Field lookup

$$\mathit{fields}(\mathtt{Object}) = \bullet \qquad \frac{\mathtt{class\ C\ extends\ D\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{V}}\ \overline{\mathtt{g}}}{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{V}}\ \overline{\mathtt{g}},\ \overline{\mathtt{U}}\ \overline{\mathtt{f}}}$$

Method type lookup

$$\frac{\begin{array}{l}\mathtt{class\ C\ extends\ D\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}} \\ \mathtt{T\ m(\overline{T}\ \overline{x})\{\ return\ e;\ \}} \in \overline{\mathtt{M}}\end{array}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{T}} \to \mathtt{T}} \qquad \frac{\mathtt{class\ C\ extends\ D\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{m} \notin \overline{\mathtt{M}}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}) = \mathit{mtype}(\mathtt{m}, \mathtt{D})}$$

Method body lookup

$$\frac{\begin{array}{l}\mathtt{class\ C\ extends\ D\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}} \\ \mathtt{T\ m(\overline{T}\ \overline{x})\{\ return\ e;\ \}} \in \overline{\mathtt{M}}\end{array}}{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{x}}.\mathtt{e}} \qquad \frac{\mathtt{class\ C\ extends\ D\ \{\overline{U}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{m} \notin \overline{\mathtt{M}}}{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \mathit{mbody}(\mathtt{m}, \mathtt{D})}$$

fields はクラスの持つフィールドを取得する関数である. スーパクラスが持つフィールドも併せて取得する. クラス `Object` はフィールドを持たないので空列を返す. *mtype* はクラスの持つ特定のメソッドの型を取得する関数である. そのクラス自身が該当するメソッドを持たない場合はスーパクラスのメソッドを検索する. *mbody* はクラスの持つ特定のメソッドの引数と本体の組を取得する関数である. そのクラス自身が該当するメソッドを持たない場合はスーパクラスのメソッドを検索する. *mtype* と *mbody* は, 部分関数であり, あるクラスとそのスーパクラスが指定されたメソッドを持たない場合には未定義となる.

FJ の型判断は $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ の形で表され, Γ は変数から型への写像からなる環境で, $\overline{\mathtt{x}} : \overline{\mathtt{T}}$ の形をしている.

定義 4 (型付け規則). FJ の型付け規則は次のように定義される.

Expression typing

$$\Gamma \vdash_{\mathrm{FJ}} \mathtt{x} : \Gamma(\mathtt{x}) \text{ (T-Var)} \qquad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{V}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash_{\mathrm{FJ}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} <: \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{new\ C}(\overline{\mathtt{e}}) : \mathtt{C}} \text{ (T-New)}$$

$$\frac{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0.\mathtt{f}_i : \mathtt{T}_i} \text{ (T-Field)} \qquad \frac{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{T} \qquad \Gamma \vdash_{\mathrm{FJ}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} <: \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{T}} \text{ (T-Invk)}$$

$$\frac{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{(C)e}_0 : \mathtt{C}} \text{ (T-UCast)} \qquad \frac{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{(C)e}_0 : \mathtt{C}} \text{ (T-DCast)}$$

$$\frac{\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash_{\mathrm{FJ}} \mathtt{(C)e}_0 : \mathtt{C}} \text{ (T-SCast)}$$

Method typing

$$\frac{\begin{array}{c}\overline{\mathtt{x}} : \overline{\mathtt{T}}, \mathtt{this} : \mathtt{C} \vdash_{\mathrm{FJ}} \mathtt{e}_0 : \mathtt{U}_0 \qquad \mathtt{U}_0 <: \mathtt{T}_0 \qquad \mathtt{class\ C\ extends\ D\ \{...\}} \\ \text{if } \mathit{mtype}(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{S}} \to \mathtt{S}_0, \text{then } \overline{\mathtt{T}} = \overline{\mathtt{S}} \text{ and } \mathtt{T}_0 = \mathtt{S}_0\end{array}}{\mathtt{T}_0\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{\{\ return\ e}_0\mathtt{;\ \}\ OK\ IN\ C}} \text{ (T-Method)}$$

Class typing

$$\frac{\mathtt{K} = \ \mathtt{C}(\overline{\mathtt{V}}\ \overline{\mathtt{g}}, \overline{\mathtt{U}}\ \overline{\mathtt{f}})\mathtt{\{super}(\overline{\mathtt{g}})\mathtt{;\ this.}\overline{\mathtt{f}}\mathtt{=}\overline{\mathtt{f}}\mathtt{;\}} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{V}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C}}{\mathtt{class\ C\ extends\ D\ \{}\overline{\mathtt{U}}\ \overline{\mathtt{f}}\mathtt{;\ K\ }\overline{\mathtt{M}}\mathtt{\}\ OK}} \text{ (T-Class)}$$

T-New では, オブジェクトコンストラクタの引数の型として, そのクラスのフィールドの型の部分型になっているものを与えている場合のみ受理する. T-Field では, レシーバのクラスのフィールドのうち名前が一致するものがある場合にのみ, そのフィールドの型を付ける. T-Invk では, レシーバのクラスに実際にメソッドが存在し, 引数の型として部分型になっているものを与えている場合のみ, メソッドの返り値の型を付ける. T-UCast, T-DCast, T-SCast はキャストに関する規則で, それぞれアップキャスト, ダウンキャスト, 明らかに不正な “愚かなキャスト”(*stupid cast* [2]) に対応する. “愚かなキャスト” では警告を出すが, 型検査自体は受理される. 実際の Java ではこのようなキャストは型検査によりエラーになるが, FJ では, プログラム実行中のキャストが失敗した状態を表現するためにこのようなキャストも項として認めている. メソッド定義の型付け規則 T-Method では, メソッド本体の型が返り値の型の部分型になっているかどうか, スーパクラスが同名のメソッドを持つ場合に, 引数と返り値の型が一致するかどうかを検査する. これは同名のメソッドはオーバーライドになり, オーバーロードを許していないためである. また, T-Method でメソッドの本体の型判断をするときには, 引数の型と, レシーバ自身のオブジェクトを参照するための特別な変数 `this` の型の情報が環境に入っている. クラス定義の型付け規則 T-Class では, コンストラクタの引数の定義が, 実際にそのクラスが持つフィールドとスーパクラスが持つフィールドに合致しているかどうか, メソッド定義が正しいかどうかを検査する.

2.3 操作的意味論

操作的意味論を簡約規則によって与える. 1 ステップの簡約は $\mathtt{e} \longrightarrow_{\mathrm{FJ}} \mathtt{e}'$ で表し, $\longrightarrow_{\mathrm{FJ}}$ の反射的推移的閉包を $\longrightarrow^*_{\mathrm{FJ}}$ で表す. また $[\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{e}/\mathtt{y}]\mathtt{e}_0$ は, 項 $\mathtt{e}_0$ の中の $\mathtt{x}_1$, ..., $\mathtt{x}_n$, $\mathtt{y}$ の出現をそれぞれ $\mathtt{d}_1$, ..., $\mathtt{d}_n$, $\mathtt{e}$ で置き換えたものを表す.

定義 5 (簡約規則). FJ の簡約規則は次のように定義される.

Computation

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\mathtt{new\ C}(\overline{\mathtt{e}})\mathtt{.f}_i \longrightarrow_{\mathrm{FJ}} \mathtt{e}_i}\ (\text{R-Field}) \qquad \frac{\mathtt{C} <: \mathtt{D}}{\mathtt{(D)new\ C}(\overline{\mathtt{e}}) \longrightarrow_{\mathrm{FJ}} \mathtt{new\ C}(\overline{\mathtt{e}})}\ (\text{R-Cast})$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{x}}.\mathtt{e}_0}{\mathtt{new\ C}(\overline{\mathtt{e}})\mathtt{.m}(\overline{\mathtt{d}}) \longrightarrow_{\mathrm{FJ}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new\ C}(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0}\ (\text{R-Invk})$$

Congruence

$$\frac{\mathtt{e}_0 \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_0}{\mathtt{e}_0\mathtt{.f} \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_0\mathtt{.f}} \qquad \frac{\mathtt{e}_0 \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_0}{\mathtt{(C)}\mathtt{e}_0 \longrightarrow_{\mathrm{FJ}} \mathtt{(C)}\mathtt{e}'_0} \qquad \frac{\mathtt{e}_i \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_i}{\mathtt{new\ C(...,\ e}_i\mathtt{,\ ...)} \longrightarrow_{\mathrm{FJ}} \mathtt{new\ C(...,\ e}'_i\mathtt{,\ ...)}}$$

$$\frac{\mathtt{e}_0 \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_0}{\mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}}) \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_0\mathtt{.m}(\overline{\mathtt{e}})} \qquad \frac{\mathtt{e}_i \longrightarrow_{\mathrm{FJ}} \mathtt{e}'_i}{\mathtt{e}_0\mathtt{.m(...,\ e}_i\mathtt{,\ ...)} \longrightarrow_{\mathrm{FJ}} \mathtt{e}_0\mathtt{.m(...,\ e}'_i\mathtt{,\ ...)}}$$

R-Field では, フィールドアクセスを簡約する. この簡約は必ずレシーバがオブジェクトコンストラクタになった後で起こり, コンストラクタに渡された引数のうちフィールドに対応するものを取り出すことで行なわれる. R-Cast では, キャストを簡約する. オブジェクトの実際のクラスがキャスト先のクラスのサブクラスになっている場合にのみキャストが成功する. R-Invk では, メソッド呼出しを簡約する. レシーバのクラスの定義からメソッドの本体を検索し, メソッド本体中の引数の変数を与えられた引数で置き換えたものに簡約する. ことのき, 特別な変数 `this` も, レシーバそのもので置き換えられる.

名前呼出しか値呼出しかといった簡約の戦略はとくに限定されていない. 残りの規則によって簡約はどの部分項からでも行なえる.

2.4 性質

FJ での計算結果の値は部分項すべてがオブジェクトコンストラクタになっているオブジェクトである.

定義 6 (値). FJ の値は以下のように定義される.

$\mathtt{v}$ ::= $\mathtt{new\ C}(\overline{\mathtt{v}})$

FJ では, 静的型エラーがなく, 実行時にキャストに失敗しなければ, それ以上簡約できない正規形の項はすべて値となる. しかもその値の実際の型は, 簡約する前の項に静的についた型の部分型であることが保証される. このことを表す定理を以下に示す.

定理 1 (型保存 (subject reduction)). $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ かつ $\mathtt{e} \longrightarrow_{\mathrm{FJ}} \mathtt{e}'$ ならば, ある $\mathtt{T}' <: \mathtt{T}$ に対して $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e}' : \mathtt{T}'$.

定理 2 (進行 (progress)). 型の付いた項 e について,

1. e が $\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}).\mathtt{f}$ を部分項に含むならば, ある $\overline{\mathtt{T}}$ と $\overline{\mathtt{f}}$ に対して $\mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$ かつ $\mathtt{f} \in \overline{\mathtt{f}}$.
2. e が $\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}).\mathtt{m}(\overline{\mathtt{d}})$ を部分項に含むならば, ある $\overline{\mathtt{x}}$ と $\mathtt{e}_0$ に対して $\mathit{mbody}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{x}}.\mathtt{e}_0$ かつ $\#(\overline{\mathtt{x}}) = \#(\overline{\mathtt{d}})$.

定理 3 (FJ の型安全性). $\emptyset \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ かつ $\mathtt{e} \longrightarrow^*_{\mathrm{FJ}} \mathtt{e}'$ で $\mathtt{e}'$ が正規形ならば, 次のいずれかが成り立つ.

1. $\mathtt{e}'$ は, $\emptyset \vdash_{\mathrm{FJ}} \mathtt{v} : \mathtt{S}$ かつ $\mathtt{S} <: \mathtt{T}$ なる値 v である.
2. $\mathtt{e}'$ は, 部分項として $\mathtt{(D)new\ C}(\overline{\mathtt{e}})$(ただし $\mathtt{C} \not<: \mathtt{D}$) を含む.

3 FJ$^?$

FJ$^?$ は FJ に動的型?を加えた体系である. 動的型?が導入されることで FJ$^?$ は FJ よりも静的型検査が甘くなり, より柔軟にプログラムを受け入れることになる. ただし, 検査が甘くなるのは明示的に?が指定された部分だけであり, 通常のクラス名を使って型宣言がされている部分に関してはできる限り静的型検査が行なわれる.

本節では FJ$^?$ の構文と型付け規則を定義し, FJ$^?$ の型検査の特徴を説明する. 最後に FJ$^?$ は FJ の保守的拡張であること, つまり, 動的型?が出現しない完全に静的型宣言されたプログラムでは FJ と全く同じ静的型検査が行なわれることを示す.

3.1 構文と型付け規則

実用上は型宣言の省略により動的型を表すこともできるが, ここでは明示的に型名?を書くことにする.

定義 7 (型). FJ$^?$ の型はクラスあるいは動的型?である.

$\mathtt{T} \quad ::= \quad \mathtt{C} \quad | \quad \mathtt{?}$

その他の構文定義に関しては FJ と同じであり, 部分型関係, 補助関数についても FJ と同様に定義される.

動的型?を導入することの目的は, これによってどんなクラスともマッチするワイルドカードのような型を実現することである. そのために FJ でクラス名同士を部分型関係を使って比較していた部分を, 動的型?との比較も考慮した一貫性検査で置きかえる.

定義 8 (部分型 (subtyping)). 関係 $\lesssim$ を次のように定義する.

$$\mathtt{S} \lesssim \mathtt{T} \quad \text{iff} \quad \mathtt{S} = \mathtt{?} \quad \text{or} \quad \mathtt{T} = \mathtt{?} \quad \text{or} \quad \mathtt{S} <: \mathtt{T}$$

ここで $\lesssim$ が推移的ではないことに注意すること. 上で触れたように, ?型は, どんな型が要求される場面にも用いることができるため任意の C に対し $\mathtt{?} \lesssim \mathtt{C}$ と $\mathtt{C} \lesssim \mathtt{?}$ が成立することが求められる. しかし, Siek と Taha [1, 3] によっても指摘されているように, $\lesssim$ を, ?をクラス階層の最上位と最下位に置いた関係の反射的推移的閉包とすると, ?を経由して推移的に任意の型の間に $\lesssim$ 関係が成立してしまい. 型検査の健全性が失われてしまう.

これらの拡張を盛り込む形で, 型判断 $\Gamma \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ による FJ$^?$ の型付け規則を新たに定義する. 網かけした部分は FJ からの変更点・追加された規則を示している.

定義 9 (型付け規則). FJ$^?$ の型付け規則を次のように定義する.

Expression typing

$$\Gamma \vdash_{\mathrm{G}} \mathtt{x} : \Gamma(\mathtt{x})\ (\text{G-Var}) \qquad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{V}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash_{\mathrm{G}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} \lesssim \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{G}} \mathtt{new\ C}(\overline{\mathtt{e}}) : \mathtt{C}}\ (\text{G-New})$$

$$\frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0.\mathtt{f}_i : \mathtt{T}_i}\ (\text{G-Field1}) \qquad \frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : ?}{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0.\mathtt{f} : ?}\ (\text{G-Field2})$$

$$\frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{T} \qquad \Gamma \vdash_{\mathrm{G}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} \lesssim \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{T}}\ (\text{G-Invk1}) \qquad \frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : ? \qquad \Gamma \vdash_{\mathrm{G}} \overline{\mathtt{e}} : \overline{\mathtt{U}}}{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : ?}\ (\text{G-Invk2})$$

$$\frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash_{\mathrm{G}} \mathtt{(C)e}_0 : \mathtt{C}}\ (\text{G-UCast}) \qquad \frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{T} \qquad \mathtt{C} \lesssim \mathtt{T} \qquad \mathtt{C} \neq \mathtt{T}}{\Gamma \vdash_{\mathrm{G}} \mathtt{(C)e}_0 : \mathtt{C}}\ (\text{G-DCast})$$

$$\frac{\Gamma \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash_{\mathrm{G}} \mathtt{(C)e}_0 : \mathtt{C}}\ (\text{G-SCast})$$

Method typing

$$\frac{\overline{\mathtt{x}} : \overline{\mathtt{T}}, \mathtt{this} : \mathtt{C} \vdash_{\mathrm{G}} \mathtt{e}_0 : \mathtt{T}_0 \qquad \mathtt{T}_0 \lesssim \mathtt{T} \qquad \mathtt{class\ C\ extends\ D\ \{...\}} \qquad \text{if } \mathit{mtype}(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{S}} \to \mathtt{T}', \text{then } \overline{\mathtt{T}} = \overline{\mathtt{S}} \text{ and } \mathtt{T} = \mathtt{T}'}{\mathtt{T\ m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\mathtt{\{\ return\ e}_0\mathtt{;\ \}\ OK\ IN\ C}}\ (\text{G-Method})$$

Class typing

$$\frac{\mathtt{K} = \ \mathtt{C}(\overline{\mathtt{V}}\ \overline{\mathtt{g}}, \overline{\mathtt{U}}\ \overline{\mathtt{f}})\mathtt{\{super}(\overline{\mathtt{g}})\mathtt{;\ this.}\overline{\mathtt{f}}\mathtt{=}\overline{\mathtt{f}}\mathtt{;\}} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{V}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C}}{\mathtt{class\ C\ extends\ D\ \{}\overline{\mathtt{U}}\ \overline{\mathtt{f}}\mathtt{;\ K\ }\overline{\mathtt{M}}\mathtt{\}\ OK}}\ (\text{G-Class})$$

G-New と G-Invk1 は, FJ と異なり部分型関係に $\lesssim$ を用いることで, 引数の型が?を含む場合に対応している. G-Field2 と G-Invk2 は, レシーバの型が?になっている場合にフィールドアクセスとメソッド呼出しをほぼ無条件で認めるためのものである. ただし, メソッド呼出しに関しては引数の数の検査はできなくとも, 引数 $\overline{\mathtt{e}}$ 自体に型がつくことが前提になる点に注意する. これによって, レシーバの型が?の場合でも, 引数の項に型エラーがある場合は静的に検出できる. ?型へのキャストは用意されていないが, ?型から何らかのクラスへのキャストは G-DCast でダウンキャストの規則にまとめられている. また, G-Method でのオーバーライドのためのメソッドの引数と返り値の型のチェックでは, FJ と同じく $=$ を使っているため, クラス型を引数とするメソッドを動的型を引数とするメソッドでオーバーライドしたりその逆はできない. これを素朴に許してしまうと, オーバーライドを 2 度することで, 全く関係のない型のメソッドがひとつのメソッド呼び出し箇所で呼ばれる可能性が出てきてしまう. たとえば, C <: D <: E の部分型関係を持つ 3 つのクラスに, それぞれメソッド m(A x), m(? x), m(B x) が定義されているとき, 静的に型 E が付いた項をレシーバとして型 B のオブジェクトを引数としたメソッド m を呼び出している場合, レシーバの評価結果がサブクラス C のオブジェクトだった場合に, クラス A のオブジェクトを受け取るメソッドに B のオブジェクトを渡すことになってしまう.

3.2 FJ との関係

動的型?の全く現れない $\mathrm{FJ}^?$ のプログラムは FJ の構文規則に合致する. FJ の構文規則に合致するプログラムならば, FJ での型判断の導出と $\mathrm{FJ}^?$ での型判断の導出は全く同じものになる. CT のクラス定義と項 $\mathtt{e}$ が全て FJ の構文に従っていて, CT のクラス定義に型が付くことを $(CT, \mathtt{e}) \in \mathrm{FJ}$ と書く. この時以下の定理が成立する.

定理 4. $(CT, \mathtt{e}) \in \mathrm{FJ}$ とする. $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ ならば, ある $\mathtt{C}$ に対し $\mathtt{T} = \mathtt{C}$ かつ $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{C}$. また, 逆に $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{C}$ ならば, $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{C}$.

証明概略. $(CT, \mathtt{e}) \in \mathrm{FJ}$ より, $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ の導出規則の中には?を含む規則 G-Field2, G-Invk2 は現れない. また, G-Field1, G-Invk1, G-New, G-UCast, G-DCast, G-SCast の結論の型にも?は現れない. G-Method が Γ を増やす唯一の規則であるが, ここに?が現れることもない. すると G-Var で $\mathtt{x}$ の型が?になることもなく, $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ ならば $\mathtt{T} \neq ?$ である. これより, 導出規則の中の $\lesssim$ は $<:$ と一致する. ゆえに $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ の導出木と $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ の導出木は同じ形になる. 厳密には, $(\Leftarrow)$ と $(\Rightarrow)$ に関して, それぞれ $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ と $\overline{\mathtt{x}} : \overline{\mathtt{C}} \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ の導出に関する帰納法を用いればよい.

この定理より, $\mathrm{FJ}^?$ は FJ の保守的拡張であるといえる.

4 $\mathrm{FJ}^?$ から $\mathrm{FJ}_{\mathrm{refl}}$ への変換

$\mathrm{FJ}^?$ に意味論を与え型安全性を証明するために, まず FJ にリフレクションを加えた $\mathrm{FJ}_{\mathrm{refl}}$ を定義し, $\mathrm{FJ}_{\mathrm{refl}}$ の性質を示す. さらに $\mathrm{FJ}^?$ から $\mathrm{FJ}_{\mathrm{refl}}$ への変換を定義することで, $\mathrm{FJ}^?$ に型安全な意味論を間接的に与えることができる. このような変換を行なわない方法としては, たとえば?を `Object` に読み替え, `Object` 型のオブジェクトを要求される型へ適宜キャストするというものが考えられるが, このような方法では `Object` 型のレシーバをどのような型へキャストすればよいかが静的に決定できない. リフレクションを用いた方法では, レシーバの簡約結果のオブジェクトが実際に正しいフィールドやメソッドを持つかどうか実行時に検査する.

4.1 $\mathrm{FJ}_{\mathrm{refl}}$ の型付け規則と意味論

$\mathrm{FJ}_{\mathrm{refl}}$ の項は FJ の項にフィールドアクセスとメソッド呼出しに関するリフレクションを加えたものである. 実際の Java では, これらの操作は複数の操作の組み合わせで実現されるが, それぞれ単一の構文で表現する. $get(\mathtt{e}, \mathtt{f})$ は項 $\mathtt{e}$ が表すオブジェクトに対して $\mathtt{f}$ の名前を持つフィールドにアクセスする. $\mathtt{e}$ を評価した結果のオブジェクトがそのような名前のフィールドを持たなければ, 実行時エラーとなる. $invoke(\mathtt{e}, \mathtt{m}, \overline{\mathtt{e}})$ は同様に項 $\mathtt{e}$ が表すオブジェクトに対してメソッド $\mathtt{m}$ を呼出す. ただし, 引数の数が一致しない場合や, 引数を簡約した結果の値がメソッド定義の引数の部分型になっていない場合は, 実行時エラーとなる.

定義 10 (項). $\mathrm{FJ}_{\mathrm{refl}}$ の項を次のように定義する.

$$\mathtt{e} \quad ::= \quad \mathtt{x} \mid \mathtt{e.f} \mid \mathtt{e.m}(\overline{\mathtt{e}}) \mid \mathit{get}(\mathtt{e}, \mathtt{f}) \mid \mathit{invoke}(\mathtt{e}, \mathtt{m}, \overline{\mathtt{e}}) \mid \mathtt{new\ C}(\overline{\mathtt{e}}) \mid \mathtt{(C)e}$$

項を除いた構文定義は FJ と同様である. 補助関数, 部分型関係も FJ と同様に定義される.

定義 11 (型付け規則). $\mathrm{FJ}_{\mathrm{refl}}$ の型付け規則を次のように定義する.

$$\Gamma \vdash_{\mathrm{R}} \mathtt{x} : \Gamma(\mathtt{x}) \ (\text{TR-VAR}) \qquad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{V}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} <: \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C}} \ (\text{TR-NEW})$$

$$\frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0.\mathtt{f}_i : \mathtt{T}_i} \ (\text{TR-FIELD1}) \qquad \frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{T}}{\Gamma \vdash_{\mathrm{R}} \mathit{get}(\mathtt{e}_0, \mathtt{f}) : \mathtt{Object}} \ (\text{TR-FIELD2})$$

$$\frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{T} \qquad \Gamma \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} <: \overline{\mathtt{V}}}{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{T}} \ (\text{TR-INVK1}) \qquad \frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{T} \qquad \Gamma \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}}}{\Gamma \vdash_{\mathrm{R}} \mathit{invoke}(\mathtt{e}_0, \mathtt{m}, \overline{\mathtt{e}}) : \mathtt{Object}} \ (\text{TR-INVK2})$$

$$\frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash_{\mathrm{R}} (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \ (\text{TR-UCAST}) \qquad \frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash_{\mathrm{R}} (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \ (\text{TR-DCAST})$$

$$\frac{\Gamma \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash_{\mathrm{R}} (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \ (\text{TR-SCAST})$$

リフレクション呼出し項のレシーバの型は任意であり, 式全体には Object 型が与えられる (TR-FIELD2 と TR-INVK2).

定義 12 (簡約規則). $\mathrm{FJ}_{\mathrm{refl}}$ の簡約規則を次のように定義する.

Computation

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})).\mathtt{f}_i \longrightarrow_{\mathrm{R}} \mathtt{e}_i} \ (\text{RR-FIELD1}) \qquad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\mathit{get}((\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})), \mathtt{f}_i) \longrightarrow_{\mathrm{R}} \mathtt{e}_i} \ (\text{RR-FIELD2})$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{x}}.\mathtt{e}_0}{(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})).\mathtt{m}(\overline{\mathtt{d}}) \longrightarrow_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0} \ (\text{RR-INVK1})$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{x}}.\mathtt{e}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{V}} \to \mathtt{T} \qquad \#(\overline{\mathtt{x}}) = \#(\overline{\mathtt{d}})}{\mathit{invoke}((\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})), \mathtt{m}, \overline{\mathtt{d}}) \longrightarrow_{\mathrm{R}} [((\overline{\mathtt{V}})\overline{\mathtt{d}})/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0} \ (\text{RR-INVK2})$$

$$\frac{\mathtt{C} <: \mathtt{D}}{(\mathtt{D})(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})) \longrightarrow_{\mathrm{R}} \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})} \ (\text{RR-CAST})$$

部分項を簡約するための congruence 規則は省略しているが, 本質的には FJ の簡約規則に RR-FIELD2 と RR-INVK2 を加えたものである. リフレクションによるメソッド呼び出しでは, レシーバがメソッド m を持つことだけでなく, 仮引数と実引数の数が一致することが検査される. さらに, 実引数が仮引数の型の部分型になっているかを検査するキャストが挿入される. ここではキャストの構文を利用して検査を行っているが, これは, $\mathrm{FJ}_{\mathrm{refl}}$ の簡約が FJ 同様戦略を固定していないためである. 値呼びにするなら, 実引数 $\overline{\mathtt{d}}$ は全て値, つまり $\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})$ の形をしているので, キャストを使わずに, この規則で実引数のクラスが適切なものか検査することができる.

4.2 $\mathrm{FJ}_{\mathrm{refl}}$ の性質

$\mathrm{FJ}_{\mathrm{refl}}$ の簡約規則は FJ の保守的な拡張になっている.

定理 5. $(CT, \mathtt{e}) \in \mathrm{FJ}$, $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ のとき, $\mathtt{e} \longrightarrow_{\mathrm{FJ}} \mathtt{e}'$ iff $\mathtt{e} \longrightarrow_{\mathrm{R}} \mathtt{e}'$.

証明概略. 簡約規則の導出に関する帰納法による.

$\mathrm{FJ}_{\mathrm{refl}}$ の安全性に関する証明を行なう. リフレクションに関する実行時エラーもキャストの場合のそれと同じように扱うことで, $\mathrm{FJ}_{\mathrm{refl}}$ での型安全性の証明も FJ に関して示されているのと同じように示すことができる. 型安全性を示すための 2 つの定理, 型保存性と進行性を示し, 型安全性に関する定義を与えて証明を行なう.

定理 6 (型保存 (subject reduction)). $\Gamma \vdash_{\mathrm{R}} \mathtt{e} : \mathtt{T}$ かつ $\mathtt{e} \longrightarrow_{\mathrm{R}} \mathtt{e}'$ ならば, ある $\mathtt{T}' <: \mathtt{T}$ に対して $\Gamma \vdash_{\mathrm{R}} \mathtt{e}' : \mathtt{T}'$.

定理 7 (進行 (progress)). $\mathtt{e}$ を型の付いた $\mathrm{FJ}_{\mathrm{refl}}$ 項とする.

1. $\mathtt{e}$ が $\mathtt{new\ C_0(\overline{e}).f}$ を部分項に含むならば, ある $\overline{\mathtt{T}}$ と $\overline{\mathtt{f}}$ に対して $\mathit{fields}(\mathtt{C_0}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$ かつ $\mathtt{f} \in \overline{\mathtt{f}}$.
2. $\mathtt{e}$ が $\mathtt{new\ C_0(\overline{e}).m(\overline{d})}$ を部分項に含むならば, ある $\overline{\mathtt{x}}$ と $\mathtt{e_0}$ に対して $\mathit{mbody}(\mathtt{m}, \mathtt{C_0}) = \overline{\mathtt{x}}.\mathtt{e_0}$ かつ $\#(\overline{\mathtt{x}}) = \#(\overline{\mathtt{d}})$.

証明. TR-Field1 および TR-Invk1 より明らか. □

キャストの失敗による実行時のエラーを以下のように定義する.

定義 13 (Bad Cast).

$$\mathit{BadCast}\ \mathtt{e} \equiv (\mathtt{e} = \mathtt{(D)new\ C(\overline{e})}) \wedge \mathtt{C} \not<: \mathtt{D}$$

リフレクションの失敗による実行時のエラーを以下のように定義する.

定義 14 (No Such Field, No Such Method).

$$\mathit{NoSuchField}\ \mathtt{e} \equiv (\mathtt{e} = \mathit{get}(\mathtt{(new\ C(\overline{e})), x})) \wedge \neg\left(\mathit{fields}(\mathtt{C}) = \overline{\mathtt{T}}\ \overline{\mathtt{f}} \wedge \mathtt{x} = \mathtt{f}_i\right)$$

$$\mathit{NoSuchMethod}\ \mathtt{e} \equiv (\mathtt{e} = \mathit{invoke}(\mathtt{(new\ C(\overline{e})), m, \overline{d}})) \wedge \neg\left(\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{x}}.\mathtt{e_0} \wedge \#(\overline{\mathtt{x}}) = \#(\overline{\mathtt{d}})\right)$$

$\mathrm{FJ}_{\mathrm{refl}}$ での値も FJ と同様に, オブジェクトコンストラクタのみからなるオブジェクトである.

定義 15 (値). $\mathrm{FJ}_{\mathrm{refl}}$ の値は以下のように定義される.

$\mathtt{v}$ ::= $\mathtt{new\ C(\overline{v})}$

定理 8 ($\mathrm{FJ}_{\mathrm{refl}}$ の型安全性). $\emptyset \vdash_{\mathrm{R}} \mathtt{e} : \mathtt{T}$ かつ $\mathtt{e} \longrightarrow_{\mathrm{R}}^{*} \mathtt{e}'$ で $\mathtt{e}'$ が正規形ならば, 次のいずれかが成り立つ.

1. $\mathtt{e}'$ は, $\emptyset \vdash_{\mathrm{R}} \mathtt{v} : \mathtt{C}$ かつ $\mathtt{C} <: \mathtt{T}$ なる値 $\mathtt{v}$ である.
2. $\mathtt{e}'$ のある部分項 $\mathtt{e_0}$ に対して $\mathit{BadCast}\,\mathtt{e_0}$ が成り立つ.
3. $\mathtt{e}'$ のある部分項 $\mathtt{e_0}$ に対して $\mathit{NoSuchField}\,\mathtt{e_0}$ が成り立つ.
4. $\mathtt{e}'$ のある部分項 $\mathtt{e_0}$ に対して $\mathit{NoSuchMethod}\,\mathtt{e_0}$ が成り立つ.

証明. 簡約規則と定理 6,7 から明らか. □

つまり, $\mathrm{FJ}_{\mathrm{refl}}$ ではそれ以上簡約できない項は, キャストやリフレクションの失敗がない限り必ず値になるということが示された.

4.3 変換の定義と性質

項の変換で挿入されるキャストのために, 以下の補助関数を定義する.

定義 16 (補助関数).

$$\begin{aligned}\langle\!\langle \mathtt{S} \Leftarrow \mathtt{T} \rangle\!\rangle \mathtt{e} \equiv \mathrm{case}\ (\mathtt{S},\mathtt{T})\ \mathrm{of}\\ (\mathtt{?},\mathtt{C}) &\Rightarrow \mathtt{e}\\ \mid (\mathtt{C},\mathtt{?}) &\Rightarrow \mathtt{(C)e}\\ \mid (\mathtt{?},\mathtt{?}) &\Rightarrow \mathtt{e}\\ \mid (\mathtt{D},\mathtt{C}) &\Rightarrow \mathrm{if}\ \mathtt{C} <: \mathtt{D}\ \mathrm{then}\ \mathtt{e}\ \mathrm{else}\ \mathtt{(D)e}\end{aligned}$$

$\mathrm{FJ}^?$ から $\mathrm{FJ}_{\mathrm{refl}}$ への項の変換規則は, $\mathrm{FJ}^?$ の型付け規則に沿う形で与える.

定義 17 (変換). $\mathrm{FJ}^?$ から $\mathrm{FJ}_{\mathrm{refl}}$ への変換を次のように定義する.

$$\Gamma \vdash \mathtt{x} \rightsquigarrow \mathtt{x} : \Gamma(\mathtt{x}) \qquad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{V}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{e}} \rightsquigarrow \overline{\mathtt{e}'} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} \lesssim \overline{\mathtt{V}}}{\Gamma \vdash \mathtt{new\ C}(\overline{\mathtt{e}}) \rightsquigarrow \mathtt{new\ C}(\langle\!\langle \overline{\mathtt{V}} \Leftarrow \overline{\mathtt{U}} \rangle\!\rangle \overline{\mathtt{e}'}) : \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i \rightsquigarrow \mathtt{e}'_0.\mathtt{f}_i : \mathtt{T}_i} \qquad \frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{?}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathit{get}(\mathtt{e}'_0, \mathtt{f}) : \mathtt{?}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{T} \qquad \Gamma \vdash \overline{\mathtt{e}} \rightsquigarrow \overline{\mathtt{e}'} : \overline{\mathtt{U}} \qquad \overline{\mathtt{U}} \lesssim \overline{\mathtt{V}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \rightsquigarrow \mathtt{e}'_0.\mathtt{m}(\langle\!\langle \overline{\mathtt{V}} \Leftarrow \overline{\mathtt{U}} \rangle\!\rangle \overline{\mathtt{e}'}) : \mathtt{T}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{?} \qquad \Gamma \vdash \overline{\mathtt{e}} \rightsquigarrow \overline{\mathtt{e}'} : \overline{\mathtt{U}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \rightsquigarrow \mathit{invoke}(\mathtt{e}'_0, \mathtt{m}, \overline{\mathtt{e}'}) : \mathtt{?}} \qquad \frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash \mathtt{(C)e}_0 \rightsquigarrow \mathtt{(C)e}'_0 : \mathtt{C}} \qquad \frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{T} \qquad \mathtt{C} \lesssim \mathtt{T} \qquad \mathtt{C} \neq \mathtt{T}}{\Gamma \vdash \mathtt{(C)e}_0 \rightsquigarrow \mathtt{(C)e}'_0 : \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \rightsquigarrow \mathtt{e}'_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \mathit{stupid\ warning}}{\Gamma \vdash \mathtt{(C)e}_0 \rightsquigarrow \mathtt{(C)e}'_0 : \mathtt{C}}$$

$\rightsquigarrow$ の左右がそれぞれ $\mathrm{FJ}^?$ と $\mathrm{FJ}_{\mathrm{refl}}$ の項である. $\rightsquigarrow\mathtt{e}$ を除けばそのまま $\mathrm{FJ}^?$ の型付け規則になっている.

CT に関しては, メソッド定義内の項も変換し, CT 中に出現する型も変換する. メソッド定義内の項は, 引数とレシーバの型を環境 Γ として与えることで項の変換規則を用いて変換する. 型の変換は, CT 中に出現する?型をすべて `Object` 型に置き換える.

$\mathrm{FJ}^?$ で型の付いた項は, 型の付いた $\mathrm{FJ}_{\mathrm{refl}}$ の項に変換することができる.

補題 1. $(CT, \mathtt{e}) \in \mathrm{FJ}^?$ とする. このとき $\Gamma \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ iff $^{\exists}\mathtt{e}'.\ \Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$.

証明概略. ($\Rightarrow$) $\Gamma \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ の導出に関する帰納法による. ($\Leftarrow$) $\Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$ の導出に関する帰納法による.

定理 9. $(CT, \mathtt{e}) \in \mathrm{FJ}^?$ に対して $\Gamma \vdash_{\mathrm{G}} \mathtt{e} : \mathtt{T}$ ならば, $(CT, \mathtt{e})$ の変換 $(CT', \mathtt{e}') \in \mathrm{FJ}_{\mathrm{refl}}$ が存在し, Γ, $\mathtt{T}$ 中の ?を `Object` に置き換えた Γ' と $\mathtt{T}'$ に対して $^{\exists}\mathtt{T}'' <: \mathtt{T}'$ かつ $\Gamma' \vdash_{\mathrm{R}} \mathtt{e}' : \mathtt{T}''$.

証明概略. 補題 1 より $\Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$ なる $\mathtt{e}'$ が存在し, また CT 内に出現する項も同様に変換できることから CT' も存在する. $\Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$ の導出に関する帰納法より $\Gamma' \vdash_{\mathrm{R}} \mathtt{e}' : \mathtt{T}''$ が示せる.

従って, $\mathrm{FJ}^?$ を $\mathrm{FJ}_{\mathrm{refl}}$ に変換することで, 実行時のエラーが動的型に関するものと元のプログラムに書かれていたキャストの失敗だけである, という弱い意味ではあるが, 型安全な意味論を $\mathrm{FJ}^?$ に与えることができる.

さらに, FJ のプログラムに関しては, この変換は恒等変換になっており, 実行ステップも同様になる.

定理 10. $(CT, \mathtt{e}) \in \mathrm{FJ}$ で $\Gamma \vdash_{\mathrm{FJ}} \mathtt{e} : \mathtt{T}$ ならば $(CT, \mathtt{e})$ の変換 $(CT', \mathtt{e}') \in \mathrm{FJ}_{\mathrm{refl}}$ が存在し, $\Gamma \vdash_{\mathrm{R}} \mathtt{e}' : \mathtt{T}$ かつ $(CT', \mathtt{e}') = (CT, \mathtt{e})$.

証明概略. 定理 4 と補題 1 より $\Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$ なる $\mathtt{e}'$ が存在し, $\Gamma \vdash \mathtt{e} \rightsquigarrow \mathtt{e}' : \mathtt{T}$ の導出に関する帰納法において, リフレクションによるフィールドアクセスとメソッド呼出しの項への変換を導出する規則が使用されない点と, $\langle\!\langle \mathtt{S} \Leftarrow \mathtt{T} \rangle\!\rangle$ の $\mathtt{S}$, $\mathtt{T}$ がともに?になりえない点に注意する.

定理 4, 5, 10 より, FJ のように完全に型付けされた項を $\mathrm{FJ}^?$ の項として実行しても, FJ と同じだけの安全性が保証されることが分かる.

5 関連研究

プロトタイプベースのオブジェクト指向言語での漸進的型付けについての研究は, Siek と Taha [3] によって行なわれている. クラスに基づくオブジェクト指向言語では, 類似の枠組みとして Anderson と Drossopoulou [4] による BabyJ があるが, これは JavaScript に似た言語に徐々に型宣言を付加し, 完全に型宣言が行なわれたプログラムを Java のプログラムに変換して静的な型安全性を保証するものである. また BabyJ には部分型関係がない. Java への動的型の導入は Gray ら [5] によっても行なわれている. Gray らは動的型の部分を Mirror というクラスへ変換することで, リフレクションのような操作を行なっている. ただし, Gray らの研究の主眼は動的型の混在した言語の型安全性を示すことではなく, 動的型を静的型システムの言語の中で扱う方法の提案と, それを利用して動的型システムの言語と静的型システムの言語の間で互いのコードを呼び出すことである. Cartwright と Fagan [6] の Soft Typing は, 動的型言語に対して型推論をすることで型が確定する部分に関して静的に型検査を行ない, 実行時検査を減らしてパフォーマンスを向上させる. ただし, 動的型と静的型宣言を混在させるものではない. Lagorio と Zucca [7] の Just は, Java に動的型の混在を許し, リフレクションを用いるが, 型推論によって静的な安全性を保つ. 動的型の取り扱いのためにリフレクションやダウンキャストを挿入しても, それが実行時に失敗することがないようにしているため, 実行時検査に頼る場合よりも表現可能なプログラムには制限がある.

6 おわりに

6.1 まとめ

静的型システムと動的型システムの中間の枠組みである漸進的型付けは, 両者の利点を活かすものである. すなわち, 動的型を許して型を意識しない素早い開発を可能にし, 静的な型を明示した部分に関しては実行前に型エラーを検出する. 本研究では, Java を単純化した体系 FJ に漸進的型付けを導入した体系 $\mathrm{FJ}^?$ を定義し, FJ にリフレクションを加えた体系 $\mathrm{FJ}_{\mathrm{refl}}$ へ変換することで, $\mathrm{FJ}^?$ が型を明示した部分の型安全性を保つことを証明した.

6.2 今後の課題

本研究の延長上にある目標は, Java への漸進的型付けの導入である. このための大きな課題としては, ジェネリクスと動的型の混在がある. Java 5.0 ではジェネリクス [8] はクラスへの型引数情報を消去する, という変換を使って実装されているので, この方針に従うと, 動的型が型引数として使われた場合に実行時検査を行なうのが困難である. また, FJ[?] ではメソッドのオーバーライドは可能だが, オーバーロードは許していない. オーバーロードを許すようにした場合, レシーバが?型の場合にリフレクションによる解決を試みるならば, 引数の型のマッチングを工夫する必要がある.

参考文献

1. Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, September 2006.
2. Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems*, Vol. 23, No. 3, pp. 396–450, May 2001.
3. Jeremy G. Siek and Walid Taha. Gradual typing for objects. In *ECOOP'07: 21st European Conference on Object-Oriented Programming*, pp. 2–27, 2007.
4. Christopher Anderson and Sophia Drossopoulou. BabyJ - from object based to class based programming via types. In *WOOD '03*, Vol. 82. Elsevier, 2003.
5. Kathryn E. Gray, Robert Bruce Findler, and Matthew Flatt. Fine-grained interoperability through mirrors and contracts. In *OOPSLA'05: Proceedings of the 20th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, pp. 231–245, 2005.
6. R. Cartwright and M. Fagan. Soft Typing. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, pp. 278–292, 1991.
7. Giovanni Lagorio and Elena Zucca. Just: safe unknown types in Java-like languages. *Journal of Object Technology*, Vol. 6, No. 2, pp. 69–98, February 2007.
8. G. Bracha, M. Odersky, D. Stoutamire, and Philip Wadler. Making the future safe for the past: Adding Genericity to the Java Programming Language. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '98)*, pp. 183–200, October 1998.

付録

A 定理 6 (型保存性) の証明

証明に入る前に, 次の補題を示す.

補題 2 (代入における型の保存). $\Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e} : \mathtt{T}$ かつ $\overline{\mathtt{U}} <: \overline{\mathtt{V}}$ で $\Gamma \vdash_{\mathrm{R}} \overline{\mathtt{d}} : \overline{\mathtt{U}}$ ならば, ある $\mathtt{S} <: \mathtt{T}$ に対して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e} : \mathtt{S}$.

証明. $\Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e} : \mathtt{T}$ の導出に関する帰納法による.

- TR-Var のとき. $\mathtt{e} = \mathtt{x} \quad \mathtt{T} = \Gamma(\mathtt{x})$
  $\mathtt{x} \notin \overline{\mathtt{x}}$ ならば, $[\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{x} = \mathtt{x}$ より $\mathtt{S} = \mathtt{T}$. $\mathtt{x} = \mathtt{x}_i$ ならば, $\mathtt{T} = \mathtt{V}_i$ で, $[\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{x} = [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{x}_i = \mathtt{d}_i$ より $\mathtt{S} = \mathtt{U}_i$.
- TR-Field1 のとき. $\mathtt{e} = \mathtt{e}_0.\mathtt{f}_i \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{D}_0 \quad \mathit{fields}(\mathtt{D}_0) = \overline{\mathtt{S}}\ \overline{\mathtt{f}} \quad \mathtt{T} = \mathtt{S}_i$
  帰納法の仮定より, ある $\mathtt{S}_0$ が存在して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$ かつ $\mathtt{S}_0 <: \mathtt{D}_0$. 従って $\mathtt{S}_0 = \mathtt{C}_0$ となり, ある $\overline{\mathtt{T}}\ \overline{\mathtt{g}}$ に対して $\mathit{fields}(\mathtt{C}_0) = \mathit{fields}(\mathtt{D}_0), \overline{\mathtt{T}}\ \overline{\mathtt{g}}$ となるから, TR-Field1 より $\Gamma \vdash_{\mathrm{R}} ([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0).\mathtt{f}_i : \mathtt{S}_i$.

– TR-Field2 のとき. $\mathtt{e} = \mathit{get}(\mathtt{e}_0, \mathtt{f}_i) \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{S} \quad \mathtt{T} = \mathtt{Object}$
帰納法の仮定より, ある $\mathtt{S}_0$ が存在して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$. 従って TR-Field2 より $\Gamma \vdash_{\mathrm{R}} \mathit{get}([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0, \mathtt{f}_i) : \mathtt{Object}$.
– TR-Invk1 のとき. $\mathtt{e} = \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{D}_0 \quad \mathit{mtype}(\mathtt{m}, \mathtt{D}_0) = \overline{\mathtt{S}} \to \mathtt{T} \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}}$ $\overline{\mathtt{U}} <: \overline{\mathtt{S}}$
帰納法の仮定より, ある $\mathtt{C}_0$ と $\overline{\mathtt{T}}$ が存在して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{C}_0$ かつ $\mathtt{C}_0 <: \mathtt{D}_0$, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}} : \overline{\mathtt{T}}$ かつ $\overline{\mathtt{T}} <: \overline{\mathtt{U}}$. 明らかに $\mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{S}} \to \mathtt{T}$ で, $\overline{\mathtt{T}} <: \overline{\mathtt{S}}$ が成り立つから, TR-Invk1 より $\Gamma \vdash_{\mathrm{R}} ([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0).\mathtt{m}([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}}) : \mathtt{T}$.
– TR-Invk2 のとき. $\mathtt{e} = \mathit{invoke}(\mathtt{e}_0, \mathtt{m}, \overline{\mathtt{e}}) \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{T}_0 \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}}$
帰納法の仮定より, ある $\mathtt{S}_0$ と $\overline{\mathtt{T}}$ が存在して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0 : \mathtt{S}_0$, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}} : \overline{\mathtt{T}}$. TR-Invk2 より $\Gamma \vdash_{\mathrm{R}} \mathit{invoke}([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\mathtt{e}_0, \mathtt{m}, [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}}) : \mathtt{Object}$.
– TR-New のとき. $\mathtt{e} = \mathtt{new}\ \mathtt{D}(\overline{\mathtt{e}}) \quad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{S}}\ \overline{\mathtt{f}} \quad \Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}} \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{U}} \quad \overline{\mathtt{U}} <: \overline{\mathtt{S}}$
帰納法の仮定より, ある $\overline{\mathtt{T}}$ が存在して, $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}} : \overline{\mathtt{T}}$ かつ $\overline{\mathtt{T}} <: \overline{\mathtt{U}}$. $\overline{\mathtt{T}} <: \overline{\mathtt{S}}$ が成り立つから, TR-New より $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{D}([\overline{\mathtt{d}}/\overline{\mathtt{x}}]\overline{\mathtt{e}}) : \mathtt{D}$.
– TR-UCast, TR-DCast, TR-SCast のとき.
帰納法の仮定と $<:$ の推移律より明らか.

□

定理 6 (型保存性) の証明

証明. $\mathtt{e} \longrightarrow_{\mathrm{R}} \mathtt{e}'$ の導出に関する帰納法による.

– RR-Field1 のとき. $\mathtt{e} = (\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})).\mathtt{f}_i \quad \mathtt{e}' = \mathtt{e}_i \quad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$
TR-Field1 より, $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}) : \mathtt{T}_0$ かつ $\mathtt{T} = \mathtt{T}_i$. さらに TR-New より, $\Gamma \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{S}}$ かつ $\overline{\mathtt{S}} <: \overline{\mathtt{T}}$ かつ $\mathtt{T}_0 = \mathtt{C}_0$. とくに $\Gamma \vdash_{\mathrm{R}} \mathtt{e}_i : \mathtt{S}_i$. ゆえに $\mathtt{T}' = \mathtt{S}_i <: \mathtt{T}_i = \mathtt{T}$.
– RR-Field2 のとき. $\mathtt{e} = \mathit{get}(\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}), \mathtt{f}_i) \quad \mathtt{e}' = \mathtt{e}_i \quad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{T}}\ \overline{\mathtt{f}}$
TR-Field2 より, $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}) : \mathtt{T}_0$ さらに TR-New より, $\Gamma \vdash_{\mathrm{R}} \overline{\mathtt{e}} : \overline{\mathtt{S}}$ かつ $\overline{\mathtt{S}} <: \overline{\mathtt{T}}$ かつ $\mathtt{T}_0 = \mathtt{C}_0$. とくに $\Gamma \vdash_{\mathrm{R}} \mathtt{e}_i : \mathtt{S}_i$. ゆえに $\mathtt{T}' = \mathtt{S}_i <: \mathtt{T}_i = \mathtt{T}$.
– RR-Invk1 のとき. $\mathtt{e} = (\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})).\mathtt{m}(\overline{\mathtt{d}}) \quad \mathit{mbody}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{x}}.\mathtt{e}_0 \quad \mathtt{e}' = [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0$
TR-Invk1 と TR-New より, $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}) : \mathtt{C}_0$ かつ $\mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{S}$ かつ $\Gamma \vdash_{\mathrm{R}} \overline{\mathtt{d}} : \overline{\mathtt{U}}$ かつ $\overline{\mathtt{U}} <: \overline{\mathtt{V}}$. *mbody*, *mtype* の定義から明らかに, $\mathtt{C}_0 <: \mathtt{D}_0$, $\mathtt{S}' <: \mathtt{S}$ に対して $\Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}}, \mathtt{this} : \mathtt{D}_0 \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{S}'$. 従って補題 2 より, $\mathtt{T}' <: \mathtt{S}'$ に対して $\Gamma \vdash_{\mathrm{R}} [\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0 : \mathtt{T}'$.
– RR-Invk2 のとき. $\mathtt{e} = \mathit{invoke}(\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}), \mathtt{m}, \overline{\mathtt{d}}) \quad \mathit{mbody}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{x}}.\mathtt{e}_0 \quad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{V}} \to \mathtt{T}$ $\mathtt{e}' = [((\overline{\mathtt{V}})\overline{\mathtt{d}})/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0$
TR-Invk2 と TR-New より, $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}) : \mathtt{C}_0$ かつ $\Gamma \vdash_{\mathrm{R}} (\overline{\mathtt{V}})\overline{\mathtt{d}} : \overline{\mathtt{V}}$. *mbody*, *mtype* の定義から明らかに, $\mathtt{C}_0 <: \mathtt{D}_0$, $\mathtt{S}' <: \mathtt{T}$ に対して $\Gamma, \overline{\mathtt{x}} : \overline{\mathtt{V}}, \mathtt{this} : \mathtt{D}_0 \vdash_{\mathrm{R}} \mathtt{e}_0 : \mathtt{S}'$. 従って補題 2 より, ある $\mathtt{T}' <: \mathtt{S}'$ に対して $\Gamma \vdash_{\mathrm{R}} [((\overline{\mathtt{V}})\overline{\mathtt{d}})/\overline{\mathtt{x}}, \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})/\mathtt{this}]\mathtt{e}_0 : \mathtt{T}'$.
– RR-Cast のとき. $\mathtt{e} = (\mathtt{D})(\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})) \quad \mathtt{C}_0 <: \mathtt{D} \quad \mathtt{e}' = \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})$
TR-UCast と TR-New より $\mathtt{T} = \mathtt{D}$ かつ $\Gamma \vdash_{\mathrm{R}} (\mathtt{D})(\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}})) : \mathtt{D}$ かつ $\Gamma \vdash_{\mathrm{R}} \mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}) : \mathtt{C}_0$. ゆえに $\mathtt{T}' = \mathtt{C}_0$.

この他の簡約規則に関しては, 帰納法の仮定と $<:$ の推移律を用いれば明らか. □