Java Stream APIによるストリーム操作の停止性検査のための型システム

A Type System for Detecting Non-Terminating Stream Operations with Java Stream API

長谷川 健太 * **桑原 寛明** † **國枝 義敏** ‡

Summary. In this paper, we propose a type system to detect non-terminating stream operations in Java programs using Stream API. Java Stream API makes it possible to write programs processing data collections in a declarative manner. Although Java Stream API has many stream operations and these operations are able to be applied to both finite and infinite streams, some operations on an infinite stream do not or may not terminate. Our proposed type system can detect these non-terminating operations. We prove the soundness of our type system and implement a type checker based on our type system with Checker Framework.

1 はじめに

Java Stream API（以下，単に Stream API）は，コレクションなどに含まれるデータ群をストリームとして扱うことで処理を行う API である．Stream API を用いることでデータ列に対する繰り返し処理や条件分岐処理，並列処理を簡潔に記述できる．Stream API は，データ群からストリームを生成する生成操作，ストリーム中の各要素を加工して新たなストリームに変換する中間操作，ストリーム中の各要素から何らかの結果を生成してストリーム操作を終了する終端操作の 3 種類の操作に分類できる．生成操作で生成されるストリームには，ストリーム中の要素数が有限個のストリーム（以下，有限ストリーム）と無限個のストリーム（以下，無限ストリーム）が存在する．Stream API はこれらのストリームを区別しないが，一部の中間操作と終端操作は無限ストリームに対して適用すると処理が停止しない．そのため，正常にコンパイルできるプログラムであっても実行が停止しないなどの予期せぬ事態が発生する可能性がある．無限ストリームに対する停止しない操作の適用をプログラムの実行前に静的に検出できることが望ましい．

本稿では，無限ストリームに対する停止しない操作の適用を検出する型システムを提案する．型システムを構築するために，Java の型システムのモデル化に用いられる Featherweight Java（以下，FJ）[1] に Stream API の要素を追加した言語 FJ^s を定義する．FJ^s の型として，無限ストリームを表す Inf 型と有限ストリームを表す Fin 型を定義し，これらの型に基づく型付け規則を定義する．型付け可能な FJ^s プログラムでは，無限ストリームに対して停止しない操作が適用されないことが保証される．提案する型システムを，Java プログラム向けの型検査器を作成するためのフレームワークである Checker Framework [2] を用いて実装し，簡単なプログラムに対して無限ストリームに対する停止しない操作の適用を検出できる例を示す．

本稿の構成は以下の通りである．2 章で Java Stream API とその問題点について述べ，3 章で Stream API によるストリーム操作の停止性を検査する型システムを提案し，健全性を示す．4 章で提案する型システムを実現する型検査器の実装について述べ，型検査器の適用例を示す．5 章で関連研究を挙げ，6 章でまとめる．

*Kenta Hasegawa, 立命館大学大学院情報理工学研究科

†Hiroaki Kuwabara, 南山大学情報センター

‡Yoshitoshi Kunieda, 立命館大学情報理工学部

2 Java Stream API

2.1 概要

Java Stream API は，コレクションなどに含まれるデータ群をストリームとして扱って処理を行う API であり，Java 8 で標準ライブラリに導入された．Stream API は主に `java.util.stream` パッケージの `Stream<T>`，`IntStream`，`DoubleStream`，`LongStream` インタフェースで構成される．各インタフェースはそれぞれ型引数 `T` が表すオブジェクト参照型，`int`，`double`，`long` 型の値からなるストリームを表す．

Stream API の各インタフェースが持つ様々なメソッドは，ストリームの生成操作，ストリーム中の各要素を加工して新たなストリームに変換する中間操作，ストリーム中の各要素から何らかの結果を生成してストリーム操作を終了する終端操作の 3 種類に分類できる．Stream API による一連のストリーム操作は，最初に 1 回の生成操作，次に 0 回以上の中間操作，最後に 1 回の終端操作からなる．Stream API は遅延評価に基づいており，中間操作の適用時点では処理は行われず，終端操作が適用された時点で全体の処理が開始される．プログラム中の記述としては一連のストリーム操作によりストリームが順次変換されるように見えるが，実際には，ストリーム中のある要素に対してすべての中間操作と終端操作を適用してから次の要素に対して再度すべての操作を適用するというように処理が進行する．

2.2 生成操作

生成操作に分類されるメソッドの一覧を表 1 に示す．表 1 の 3 列目は，生成されるストリームが有限であるか無限であるかを表す．`of`，`ofNullable`，`stream`，`empty`，`range` は有限ストリーム，`generare`，`iterate` は無限ストリームを生成する．2 つのストリームを連結する `concat` も生成操作の一種だが，生成されるストリームが連結対象のストリームに依存して有限にも無限にもなり得るため本稿では扱わない．

2.3 中間操作

中間操作に分類されるメソッドの一覧を表 2 に示す．`map` と同様の処理を行う `mapToInt`，`mapToDouble`，`mapToLong` は省略している．表 2 の 3 列目は，無限ストリームに対して適用した場合に，1 つ以上かつ有限個の要素を消費して操作が完了し停止するか否かを表す．停止すれば次の操作の適用に進む．表 2 の 4 列目は，無限ストリームに対して適用した結果返されるストリームが有限であるか無限であるかを表す．`sorted` は無限個の要素をソートできないため停止せずストリームを返さない[1]．`distinct`，`filter`，`dropWhile` は適切な要素を発見できるまでストリームを走査するため，無限ストリーム中に適切な要素が存在しない場合は停止しない．`map`，`skip`，`peek`，`takeWhile` はストリーム中の 1 要素から結果を生成するため常に停止し，任意の要素に適用できるため無限ストリームを返す．`limit` はストリーム

表 1　生成操作のメソッド一覧

メソッド名	説明	結果
`of`	指定された要素からなるストリームを生成	有限
`ofNullable`	指定された単一の要素からなるストリームを生成	有限
`stream`	データ集合からストリームを生成	有限
`empty`	空のストリームを生成	有限
`range`	指定された範囲の値からなるストリームを生成	有限
`generate`	固定値の無限ストリームを生成	無限
`iterate`	指定された値を基点に無限ストリームを生成	無限

[1] 実際には OutOfMemoryError で停止するがこれは操作の正常な完了ではない．

表 2　中間操作のメソッド一覧

メソッド名	説明	停止性	結果
sorted	自然順序に従いソート	停止しない	—
distinct	重複する値の削除	場合による	無限
filter	条件に基づくフィルタリング	場合による	無限
dropWhile	先頭要素から条件を満たす間除外	場合による	無限
map	条件に基づく要素の変換	停止する	無限
skip	ストリーム先頭要素の個数指定除外	停止する	無限
peek	指定された処理を実行	停止する	無限
takeWhile	先頭要素から条件を満たす間取得	停止する	無限
limit	ストリーム長の制限	停止する	有限

表 3　終端操作のメソッド一覧

メソッド名	説明	停止性
forEach	繰り返し操作	停止しない
count	ストリーム中の要素数の返却	停止しない
min	ストリーム中の要素の最小値の返却	停止しない
max	ストリーム中の要素の最大値の返却	停止しない
reduce	要素の集約	停止しない
collect	可変なオブジェクトに対する簡約操作	停止しない
toArray	ストリームの要素からなる配列の返却	停止しない
anyMatch	条件を満たす要素が存在するか判定	場合による
allMatch	全要素が条件を満たすか判定	場合による
noneMatch	全要素が条件を満たさないか判定	場合による
findFirst	ストリームの先頭要素の返却	停止する
findAny	ストリームの任意要素の返却	停止する

の先頭から指定された数の要素を返すため結果は有限ストリームとなる．表 2 のメソッドを有限ストリームに対して適用した場合は常に停止し有限ストリームを返す．

ストリーム中の各要素からストリームを生成して連結する flatMap も中間操作に分類できる．map と同様に 1 要素から結果を生成するため常に停止するが，返されるストリームは各要素から生成されるストリームに依存して有限にも無限にもなり得る．flatMap は有限ストリームに対する動作が異なるため本稿では扱わない．

2.4　終端操作

終端操作に分類されるメソッドの一覧を表 3 に示す．forEachOrdered は forEach の特殊な場合であるため省略している．表 3 の 3 列目に無限ストリームに対して適用した場合の停止性を示す．forEach，count，min，max，reduce，collect，toArray はストリーム中の全要素を走査するため停止しない．anyMatch，allMatch，noneMatch は，指定した条件を満たすあるいは満たさない要素がストリーム中に存在する場合は停止し，存在しない場合は停止しない．findFirst はストリーム中の先頭要素を，findAny はストリーム中の任意の要素を返すため停止する．有限ストリームに適用した場合はすべての終端操作が停止する．

2.5　Stream API の問題点

Stream API には，無限ストリームに対して適用すると停止しない中間操作と終端操作が存在する．このような操作がプログラム中に存在しても検出する手段がなく，ストリーム操作が停止するかしないかは実際に実行されるまでわからない．

2.5.1 停止しない中間操作

sorted はソートを行うためにストリーム中のすべての要素を必要とするが，無限ストリームは要素が無限に存在するため操作が完了しない．filter はストリームから指定された条件を満たす要素を選別する操作であるが，条件を満たす要素が見つかるまでストリームを走査し続けるため，無限ストリームに条件を満たす要素が存在しなければ操作が完了しない．distinct と dropWhile も同様である．中間操作が完了しない場合，次の中間操作あるいは終端操作の適用へと処理が進まない．

無限ストリームに対して sorted を適用するプログラム例をソースコード 1 に示す．このプログラムは，iterate を用いて 0 以上の整数の無限ストリームを生成し，sorted によりストリーム中の全要素をソートし，limit によりストリームの先頭 10 個の要素を含む有限ストリームに変換し，最後にストリーム中の要素を標準出力する．無限ストリームを有限ストリームに変換する limit が適用されるため終端操作の forEach は停止することが期待されるが，limit の前の sorted が無限ストリームに適用されるために停止せず，limit の適用まで処理が進行しない．

ソースコード 1　停止しない中間操作の例

```
Stream.iterate(0,i->i+1).sorted().limit(10).forEach(System.out::println);
```

2.5.2 停止しない終端操作

ストリーム中のすべての要素を走査する終端操作を無限ストリームに対して適用すると，無限個の要素を処理するため操作が完了せず，結果として実行が停止しない．

プログラムの例をソースコード 2 に示す．このプログラムは，iterate を用いて生成される無限ストリームに対して forEach を適用してストリーム中の全要素を標準出力する．iterate は要素を無限に生成するため，forEach は停止しない．

ソースコード 2　停止しない終端操作の例

```
Stream.iterate(0,i->i+1).forEach(System.out::println);
```

3　型システム

Stream API によるストリーム操作の停止性を検査する型システムを提案する．FJ [1] からキャストを除き Stream API の要素を導入した言語 FJ^s に対して型付け規則を定義し，ストリーム操作の停止性に関する健全性，すなわち型付け可能な FJ^s プログラムでは無限ストリームに対して停止しない操作が適用されないことを示す．

3.1　FJ^s と Stream API との対応関係

生成，中間，終端操作に分類できる Stream API を，さらに操作結果のストリームの有限性および無限ストリームに対する停止性に着目して以下の 7 種類に分類する．

- 有限ストリームの生成操作
- 無限ストリームの生成操作
- 無限ストリームに対して停止しない中間操作
- 無限ストリームに対して停止し，有限ストリームに変換しない中間操作
- 無限ストリームに対して停止し，有限ストリームに変換する中間操作
- 無限ストリームに対して停止しない終端操作
- 無限ストリームに対して停止する終端操作

有限ストリームの生成操作として fin，無限ストリームの生成操作として inf，その他 5 種類の操作それぞれの代表として $sorted$，map，$limit$，$forEach$，$findFirst$ を FJ^s に導入する．$sorted$，map，$limit$，$forEach$，$findFirst$ はそれぞれ Stream API の同名のメソッドに対応する．中間操作の distinct，filter，dropWhile については，FJ^s として形式化する上では，停止する場合は map，停止しない場合は $sorted$ とみなすことができるため，FJ^s に対応する操作を導入しない．終端操作の anyMatch，allMatch，noneMatch についても，形式化する上では停止する場合は

$$
\begin{aligned}
CL &::= \text{class}\, C\, \text{extends}\, C\{\overline{\tau}\,\overline{f};\ K\ \overline{M}\}\\
K &::= C(\overline{\tau}\,\overline{f})\{\text{super}(\overline{f});\ \text{this}.\overline{f} = \overline{f};\}\\
M &::= \tau\, m(\overline{\tau}\,\overline{x})\{\text{return}\, t;\}\\
\tau &::= C \mid Unit \mid Inf \mid Fin\\
t &::= x \mid t.f \mid t.m(\overline{t}) \mid \text{new}\, C(\overline{t}) \mid unit \mid stream\\
stream &::= st \mid st.findFirst \mid st.forEach\\
st &::= st.sorted \mid st.map \mid st.limit \mid inf \mid fin
\end{aligned}
$$

図 1　FJ^s の構文

$$
fields(\text{Object}) = \bullet \qquad \frac{\text{class}\, C\, \text{extends}\, D\{\overline{\psi}\,\overline{f};\ K\ \overline{M}\} \qquad fields(D) = \overline{\phi}\,\overline{g}}{fields(C) = \overline{\phi}\,\overline{g}, \overline{\psi}\,\overline{f}}
$$

$$
\frac{\text{class}\, C\, \text{extends}\, D\{\overline{\psi}\,\overline{f};\ K\ \overline{M}\} \qquad \tau m(\overline{\tau}\,\overline{x})\{\text{return}\, t;\} \in \overline{M}}{mtype(m,\, C) = \overline{\tau} \to \tau}
$$

$$
\frac{\text{class}\, C\, \text{extends}\, D\{\overline{\psi}\,\overline{f};\ K\ \overline{M}\} \qquad m \notin \overline{M}}{mtype(m,\, C) = mtype(m,\, D)}
$$

$$
\frac{\text{class}\, C\, \text{extends}\, D\{\overline{\psi}\,\overline{f};\ K\ \overline{M}\} \qquad \tau\, m(\overline{\tau}\,\overline{x})\{\text{return}\, t;\} \in \overline{M}}{mbody(m,\, C) = \overline{x}.t}
$$

$$
\frac{\text{class}\, C\, \text{extends}\, D\{\overline{\psi}\,\overline{f};\ K\ \overline{M}\} \qquad m \notin \overline{M}}{mbody(m,\, C) = mbody(m,\, D)}
$$

図 2　補助関数

$findFirst$，停止しない場合は $forEach$ とみなせばよい．

3.2　構文と補助関数

FJ^s の構文を図 1 に示す．以下，メタ変数 C，D，E はクラス名，τ，ψ，ϕ は型名，f，g はフィールド名，m はメソッド名，x は仮引数名を表す．$\overline{f}$ は $f_1, \ldots, f_n$ の略記，$\overline{M}$ は $M_1 \ldots M_n$ の略記，$\overline{\tau}\,\overline{f}$; は $\tau_1 f_1; \ldots \tau_n f_n;$ の略記，$\overline{\tau}\,\overline{f}$ は $\tau_1 f_1, \ldots, \tau_n f_n$ の略記，$\text{this}.\overline{f} = \overline{f};$ は $\text{this}.f_1 = f_1; \ldots \text{this}.f_n = f_n;$ の略記である．列 $\overline{f}$ の長さを $\#(\overline{f})$ と表す．f が $\overline{f}$ に含まれることを $f \in \overline{f}$ と表す．

クラス定義 CL は，親クラスの指定，フィールド宣言，コンストラクタ宣言 K，メソッド宣言列 $\overline{M}$ を含む．同一クラス内でフィールド名，メソッド名の重複はなく，親クラスが持つフィールドと同名のフィールドは宣言できないとする．super は親クラスのフィールドを初期化するコンストラクタ，this はレシーバオブジェクトを参照する変数である．コンストラクタは，クラスが持つ各フィールドに対応する引数を受け取り，各フィールドを対応する引数の値で初期化する．メソッド宣言中の τ は戻り値の型，$\overline{\tau}\,\overline{x}$ は仮引数の型と変数名，t はメソッドが返す式である．

型 τ はクラス C，$Unit$，Inf，Fin である．$Unit$ 型は Java の void 型に相当し，$unit$ が唯一の値である．Inf，Fin 型はそれぞれ無限，有限のストリームを表す．

式 t は，変数参照 x，フィールドアクセス $t.f$，メソッド呼び出し $t.m(\overline{t})$，オブジェクト生成 $\text{new}\, C(\overline{t})$，$unit$，ストリーム操作 $stream$ である．ストリーム操作はストリーム st に対する終端操作 $findFirst$，$forEach$ の適用 $st.findFirst$，$st.forEach$ である．ストリームは中間操作 $sorted$，map，$limit$ の適用 $st.sorted$，$st.map$，$st.limit$，あるいは無限ストリームの生成式 inf，有限ストリームの生成式 fin である．

FJ^s の補助関数を図 2 に示す．$fields(C)$ は，クラス C とその親クラスで定義される全フィールドの型と名前の列を返す．$mtype(m, C)$ と $mbody(m, C)$ は，それぞれクラス C のメソッド m の仮引数の型の列と戻り値の型の組，仮引数の列と本体

$$\frac{}{C <: C}\ \text{[T-Refl]} \qquad \frac{\text{class } C \text{ extends } D\{\cdots\}}{C <: D}\ \text{[T-Extends]}$$

$$\frac{C <: D \quad D <: E}{C <: E}\ \text{[T-Trans]} \qquad \frac{}{Unit <: Unit}\ \text{[T-UnitRefl]}$$

$$\frac{}{Fin <: Fin}\ \text{[T-FinRefl]} \qquad \frac{}{Inf <: Inf}\ \text{[T-InfRefl]} \qquad \frac{}{Fin <: Inf}\ \text{[T-Stream]}$$

図 3　部分型規則

$$\frac{}{\Gamma \vdash x\colon \Gamma(x)}\ \text{[T-Var]} \qquad \frac{\Gamma \vdash t_0\colon C_0 \qquad fields(C_0) = \overline{\tau}\ \overline{f}}{\Gamma \vdash t_0.f_i\colon \tau_i}\ \text{[T-Field]}$$

$$\frac{\Gamma \vdash t_0\colon C_0 \qquad \Gamma \vdash \overline{t}\colon \overline{\psi} \qquad mtype(m, C_0) = \overline{\phi} \to \tau \qquad \overline{\psi} <: \overline{\phi}}{\Gamma \vdash t_0.m(\overline{t})\colon \tau}\ \text{[T-Invoke]}$$

$$\frac{fields(C) = \overline{\phi}\ \overline{f} \qquad \Gamma \vdash \overline{t}\colon \overline{\psi} \qquad \overline{\psi} <: \overline{\phi}}{\Gamma \vdash \text{new } C(\overline{t})\colon C}\ \text{[T-New]} \qquad \frac{}{\Gamma \vdash unit\colon Unit}\ \text{[T-Unit]}$$

$$\frac{}{\Gamma \vdash inf\colon Inf}\ \text{[T-Inf]} \qquad \frac{}{\Gamma \vdash fin\colon Fin}\ \text{[T-Fin]} \qquad \frac{\Gamma \vdash st\colon Fin}{\Gamma \vdash st.sorted\colon Fin}\ \text{[T-Sorted]}$$

$$\frac{\Gamma \vdash st\colon \tau \quad \tau \in \{Fin, Inf\}}{\Gamma \vdash st.map\colon \tau}\ \text{[T-Map]} \qquad \frac{\Gamma \vdash st\colon \tau \qquad \tau \in \{Fin, Inf\}}{\Gamma \vdash st.limit\colon Fin}\ \text{[T-Limit]}$$

$$\frac{\Gamma \vdash st\colon Fin}{\Gamma \vdash st.forEach\colon Unit}\ \text{[T-ForEach]} \qquad \frac{\Gamma \vdash st\colon \tau \qquad \tau \in \{Fin, Inf\}}{\Gamma \vdash st.findFirst\colon Unit}\ \text{[T-FindFirst]}$$

図 4　式の型付け規則

の式の組を返す．いずれも m が C で宣言されていなければ親クラス D を検索する．

3.3　型付け規則

FJ^s の部分型規則を図 3 に示す．T-Refl は部分型の反射律，T-Extends はクラスの継承関係，T-Trans は部分型の推移律を表す．T-UnitRefl，T-FinRefl，T-InfRefl はそれぞれ $Unit$，Fin，Inf 型の部分型関係における反射律を表す．T-Stream は Inf 型と Fin 型の関係を表し，Fin 型の式に Inf 型の式が代入できないことを示す．

FJ^s の式の型付け規則を図 4 に示す．環境 Γ は，変数から型への有限の写像であり，$\overline{t}\colon \overline{\tau}$ で表す．$\Gamma \vdash \overline{t}\colon \overline{\tau}$ は $\Gamma \vdash t_1\colon \tau_1, ..., \Gamma \vdash t_n\colon \tau_n$ の略記であり，$\overline{C} <: \overline{D}$ は，$C_1 <: D_1, ..., C_n <: D_n$ の略記である．T-Var は変数参照 x，T-Field はフィールドアクセス $t_0.f_i$，T-Invoke はメソッド呼び出し $t_0.m(\overline{t})$，T-New はオブジェクト生成式 new $C(\overline{t})$ の型付け規則である．T-Inf，T-Fin はそれぞれストリーム生成式 inf，fin の型付け規則である．T-Sorted は $st.sorted$ の型付け規則である．$sorted$ は無限ストリームに対して停止しないため，st が Fin 型の場合のみ型が付けられる．T-Map は $st.map$ の型付け規則である．map はストリームの有限性，無限性を変更しない中間操作であるため，st の型が Fin か Inf であれば，$st.map$ は st の型で型付けされる．T-Limit は $st.limit$ の型付け規則である．$limit$ はストリーム長を有限長に制限する中間操作であるため，st の型が Fin か Inf であれば，$st.limit$ は Fin 型で型付けされる．T-FindFirst は $st.findFirst$ の型付け規則である．$findFirst$ は先頭の要素を返して停止する終端操作であるため，st の型が Fin か Inf であれば，$st.findFirst$ は $Unit$ 型で型付けされる．返す値の具体的な型は重要ではないので，$st.findFirst$ の型を $Unit$ 型としている．T-ForEach は $st.forEach$ の型付け規則である．$forEach$ は無限ストリームに対して停止しないため，st が Fin 型の場合のみ型が付けられる．$st.forEach$ は値を返さないため，型を $Unit$ 型とする．

FJ^s のメソッド宣言とクラス定義の型付け規則は FJ と同じであるため省略する．

$$\frac{t_0 \to t_0'}{t_0.f \to t_0'.f} \qquad \frac{fields(C) = \overline{\tau}\ \overline{f}}{(\text{new } C(\overline{t})).f_i \to t_i}\ \text{[E-Proj]} \qquad \frac{t_0 \to t_0'}{t_0.m(\overline{t}) \to t_0'.m(\overline{t})}$$

$$\frac{t_i \to t_i'}{t_0.m(\dots,t_i,\dots) \to t_0.m(\dots,t_i',\dots)} \qquad \frac{t_i \to t_i'}{\text{new } C(\dots,t_i,\dots) \to \text{new } C(\dots,t_i',\dots)}$$

$$\frac{mbody(m,C) = \overline{x}.t_0}{(\text{new } C(\overline{t})).m(\overline{d}) \to [\overline{x} \mapsto \overline{d}, \text{this} \mapsto \text{new } C(\overline{t})]t_0}\ \text{[E-Invoke]} \qquad \frac{}{fin.sorted \to fin}$$

$$\frac{}{fin.forEach \to unit} \qquad \frac{st \to st'}{st.forEach \to st'.forEach} \qquad \frac{st \to st'}{st.sorted \to st'.sorted}$$

$$\frac{}{inf.findFirst \to unit} \qquad \frac{}{fin.findFirst \to unit} \qquad \frac{st \to st'}{st.findFirst \to st'.findFirst}$$

$$\frac{}{inf.map \to inf} \qquad \frac{}{fin.map \to fin} \qquad \frac{st \to st'}{st.map \to st'.map}$$

$$\frac{}{inf.limit \to fin} \qquad \frac{}{fin.limit \to fin} \qquad \frac{st \to st'}{st.limit \to st'.limit}$$

図 5　簡約規則

3.4　意味論

FJ^s の操作的意味論を図 5 に示す小ステップの簡約規則で与える．以降で参照する規則のみ規則名を示している．1 ステップの簡約を $t \to t'$ と表し，$\to$ の反射的推移的閉包を $\to^*$ と表す．$[\overline{x} \mapsto \overline{d}, \text{this} \mapsto \text{new } C(\overline{t})]t_0$ は，式 t_0 中の $x_1, x_2, \dots, x_n$ を $d_1, d_2, \dots, d_n$ で置換し，this を new $C(\overline{t})$ で置換した式を表す．$forEach$ と $sorted$ は無限ストリームに対して適用すると停止しないため，有限ストリーム fin に対して適用する場合のみ簡約規則を定義する．FJ^s の意味論では Stream API の停止しない呼び出しを簡約の不正な停止として表現する．

3.5　型システムの健全性

簡約規則から分かるように，無限ストリームに対する $forEach$ と $sorted$ の適用は簡約できない．型付け可能な FJ^s プログラムは，このような簡約できないストリーム操作を含まない．以下では，提案する型システムが健全であること，すなわち型付け可能な FJ^s プログラムは最終的な値まで正しく簡約できることを示す．

FJ^s における値と正規形を以下のように定義する．

定義 1. 値 v を以下のように定義する．

$$v \quad ::= \quad \text{new } C(\overline{v}) \mid unit \mid inf \mid fin$$

定義 2. 簡約できない式は正規形である．

この時，型付け可能な正規形は値に限られることが示される．

定理 1. 型付け可能な正規形は値である． □

FJ^s の型システムは停止性に関して健全であり，型付け可能な式は値に到達するまで簡約できる．健全性は以下に示す進行性と保存性から証明できる．

定理 2 (進行性)**.** t を型付け可能な式とする．

1. t が部分式として new $C_0(\overline{t}).f$ を持つならば，$fields(C_0) = \overline{\tau}\ \overline{f}$ なる $\overline{\tau}$ と $\overline{f}$ が存在して $f \in \overline{f}$.
2. t が部分式として new $C_0(\overline{t}).m(\overline{d})$ を持つならば，$mbody(m, C_0) = \overline{x}.t_0$ なる $\overline{x}$ と t_0 が存在して $\#(\overline{x}) = \#(\overline{d})$.
3. t が部分式として $st.sorted$，$st.forEach$ を持つならば，$st \to^* fin$. □

定理 3 (保存性)**.** $\Gamma \vdash t : \tau$ かつ $t \to t'$ ならば，ある $\tau' <: \tau$ に対して $\Gamma \vdash t' : \tau'$. □

進行性は，型付け可能な式に含まれるフィールドアクセス，メソッド呼び出し，ストリーム操作は簡約できることを表す．フィールドアクセスの場合，規則 T-Field からレシーバオブジェクトはアクセスされるフィールドを持っており，規則 E-Proj に

より簡約できる．メソッド呼び出しの場合，規則 T-Invoke から呼び出されるメソッドの仮引数と実引数の個数は等しく，規則 E-Invoke により簡約できる．$st.sorted$ と $st.forEach$ の st の型はそれぞれ規則 T-Sorted と規則 T-ForEach から Fin であり，この場合 $st \to^* fin$ であることは容易に示される．よって，$st.sorted$ と $st.forEach$ はともに簡約規則に従って簡約できる．保存性は，型付け可能な式を 1 ステップ簡約すると，簡約後の式も適切な型で型付け可能であることを示している．

定理 4 (健全性)．$\emptyset \vdash t\colon \tau$ の時，t' を正規形として $t \to^* t'$ ならば，t' は以下のいずれかを満たす．

1. $\tau = C$ ならば $D <: C$ なる D が存在して $\emptyset \vdash v\colon D$ となる値 v である．
2. 値 $unit$，inf，fin のいずれかである． □

4 型検査器の実装

提案する型システムに基づく型検査器を Checker Framework（以下，CF）を用いて実装した．型検査器はコンパイラにプラグインされてコンパイル時に起動される．

4.1 Checker Framework

CF は，Java に対して型システムに基づく型検査器を構築するためのフレームワークであり，型を型アノテーションとして定義し，型付け規則を構文木を走査する Visitor として定義する．ライブラリなど既存のプログラムに型を付与するために，型アノテーションで注釈されたメソッドシグネチャが含まれる stub ファイルを作成する．型検査器を利用したい開発者は，必要な型を型アノテーションを利用してプログラム中に記述する．Java コンパイラに対して利用したい型検査器をプラグインすることで，Java コンパイラによる型検査と独立して別の型検査が実行される．

4.2 型アノテーション

提案する型システムの型を表現するために，`@Infinite`，`@Finite`，`@Preserved` の 3 つの型アノテーションを定義する．`@Infinite`，`@Finite` アノテーションはそれぞれ無限ストリーム，有限ストリームを表す型アノテーションである．stub ファイルにおいて，`@Infinite` アノテーションは無限ストリームを生成し得るメソッドの戻り値の型に付与される．`@Finite` アノテーションは有限ストリームを生成するメソッド，無限ストリームを有限ストリームに変換するメソッドの戻り値の型，引数に有限ストリームを指定するメソッドの引数の型に付与される．`@Preserved` アノテーションは，ストリームの有限性および無限性を変更しないメソッドの戻り値の型を注釈する．FJ^s に直接対応する型は存在しないが，map 操作のように返すストリームの型が適用対象のストリームの型と同じであることを表すために必要である．

これらの型アノテーションを用いて注釈された Stream API のメソッドシグネチャを含む stub ファイルを作成する．例えば，`sorted` のメソッドシグネチャは`@Preserved Stream<T> sorted() @Finite;` のように注釈される．`@Preserved` は戻り値の型が適用対象の型と同じであること，`@Finite` は有限ストリームに対してのみ適用できることを表す．ユーザプログラムにもこれらの型アノテーションを記述できるが，記述がなければ CF が推論するため，実際には記述しなくてよい．

4.3 型検査器

型検査器は図 3，4 に示す型付け規則に基づいて実装されており，規模は 500 行程度である．Stream API のメソッドのうち，`concat`，`flatMap` 以外のメソッドに対応している．`filter` など一部のメソッドは，適用するストリーム中の要素と操作の内容によって操作が停止するか否かが決まる．しかし，FJ^s では操作の内容や対象を形式化の対象とせず，操作が停止するか否かはわかるものとして形式化している．今回の実装では単純に，`distinct`，`filter`，`dropWhile` は `sorted` とみなし，

anyMatch，allMatch，noneMatch は forEach とみなして実装することとした．無限ストリームに適用されるこれらのメソッドはストリーム中の要素や操作の内容によらず常に停止しないと判断されることになるため，停止するケースについては正しく判定できない．

4.4 適用例

無限ストリームに対して停止しない操作を適用する簡単な Java プログラムをソースコード 3 に示す．@Infinite で注釈された変数 st は全要素が整数 1 である無限ストリームを保持する．st に対して中間操作 filter を用いてストリーム中の整数 0 のみを選別し，最後に終端操作 forEach を適用して選別した要素を標準出力する．filter はストリーム中に整数 0 を探すが，全要素が整数 1 であるため停止しない．

ソースコード 3　正しく判定できる停止しない操作の例

```
@Infinite Stream<Integer> st = Stream.iterate(1, i -> i);
st.filter(i -> i == 0).limit(5).forEach(System.out::println);
```

ソースコード 3 に対応する FJ^s プログラムをソースコード 4 に示す．ここで，filter を *sorted* とみなしている．*this.st* の値は *inf* であるが *inf.sorted* に対する簡約規則は存在しないため，メソッド本体の式の簡約は $inf.sorted.limit.forEach$ までで停止する．*this.st.sorted* に対する型導出木を図 6 に示す．ここで，$\Gamma = this\colon A$ である．*this.st* の型が Inf であるため規則 T-Sorted を適用できず，*this.st.sorted* は型が付けられない．

ソースコード 4　ソースコード 3 に対応する FJ^s プログラム

```
class A extends Object {
  Inf st;
  A(Inf st) { super(); this.st = st; }
  Unit m() { return this.st.sorted.limit.forEach; }
}
new A(inf).m();
```

$$\dfrac{\dfrac{\Gamma(this) = A \qquad fields(A) = Inf\ st}{\Gamma \vdash this.st\colon Inf}\ [\text{T-Field}]}{\Gamma \vdash this.st.sorted\colon ???}\ [???]$$

図 6　*this.st.sorted* の型導出木

実装した型検査器を用いてソースコード 3 を検査した結果を図 7 に示す．このエラーは，filter メソッドが呼び出されるレシーバオブジェクトについて期待される型と実際の型が一致しないことで発生している．st に対し無限ストリームを表す@Infinite アノテーションが付与されているため st は Inf 型と判断されるが，filter メソッドのレシーバの型には有限ストリームを表す@Finite アノテーションが付与されており，Fin 型が期待されているためエラーとなる．

一方，filter の条件を `i -> i == 1` とすると，ストリーム中のすべての要素が条件を満たすので，limit により先頭から 5 つ選別し，forEach により標準出力に表示して終了する．停止するストリーム操作となっているが，対応する FJ^s プログラムは型付け不能なソースコード 4 と同じであり，停止性を正しく判定できない．

```
エラー: [method.invocation.invalid] call to filter(java.util.function.Predicate<?
  super T>) not allowed on the given receiver.
        st.filter(i -> i == 0).limit(5).forEach(System.out::println);
                  ^
  found   : @Infinite Stream
  required: @Finite Stream
エラー1個
```

図 7　型検査器が発行するエラーメッセージ

5 関連研究

IntelliJ IDEA 2018.1 には Stream API の無限ストリームを検出する機能が導入されている．この機能は，無限ストリームに対し要素数を制限する操作が適用されていないことを検出する．ソースコード 1 のような要素数を制限する操作が含まれていても停止しない場合も検出できる．本研究の提案手法と検出能力は同程度であると思われるが，健全性などは議論されていない．

FJ [1] は Java の型システムを定式化するためのサブセット言語であり，FJ を用いて Java の拡張をモデル化する研究が行われている．五十嵐らは [1] の中で FJ に総称型を導入する拡張を行っている．伊奈らは，漸進的型付け [3] を FJ に導入して形式化し，オブジェクト指向言語における漸進的型付けの基盤を構築している [4]．

CF を用いて型検査器を作成する研究が行われている．例えば，Kechagia らによる API に対する不正な値の入力の検査 [5]，Mackie らによる Java プログラムにおける符号ありあるいは符号なし整数の値に関するエラーの検査 [6]，Weitz らによる Java プログラムにおけるフォーマット文字列の不正使用の検査 [7] などが型検査器として CF を用いて実装されている．CF にも null チェックを行う Nullness Checker [8] や，エイリアス関係がないことを検査する Aliasing Checker が含まれている．

6 おわりに

本稿では，Java Stream API によるストリーム操作の停止性を検査する型システムを構築し，その健全性を証明した．無限ストリームに対する停止しない操作の適用を型検査により静的に検出できる．提案する型システムに基づく型検査器を Checker Framework を用いて実装し，簡単なプログラムに対して適用した．

今後の課題として，本稿の対象から除外した `concat` と `flatMap` に対応する必要がある．停止性がストリーム中の要素や操作の内容に依存する操作に対応することや，実装した型検査器を実際のソフトウェアのプログラムに対して適用し評価することも今後の課題である．

謝辞 本研究の一部は JSPS 科研費 JP15K00112，JP17K12666，JP18K11241 および 2018 年度南山大学パッヘ研究奨励金 I-A-2 の助成による．

参考文献

[1] Igarashi, Atsushi and Pierce, Benjamin C. and Wadler, Philip. Featherweight Java: A Minimal Core Calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.*, Vol. 23, No. 3, pp. 396–450, May 2001.

[2] Matthew M. Papi, Mahmood Ali, Telmo Luis Correa Jr, Jeff H. Perkins, and Michael D Ernst. Practical pluggable types for Java. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pp. 201–212. ACM, 2008.

[3] Jeremy G. Siek and Walid Taha. Gradual Typing for Functional Languages. In *Scheme and Functional Programming Workshop*, Vol. 6, pp. 81–92, 2006.

[4] 伊奈林太郎, 五十嵐淳. Featherweight Java のための漸進的型付け. コンピュータソフトウェア, Vol. 26, No. 2, pp. 2_18–2_40, 2009.

[5] Maria Kechagia and Diomidis Spinellis. Type Checking for Reliable APIs. In *Proceedings of the 1st International Workshop on API Usage and Evolution*, pp. 15–18. IEEE Press, 2017.

[6] Christopher A Mackie. Preventing Signedness Errors in Numerical Computations in Java. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 1148–1150. ACM, 2016.

[7] Konstantin Weitz, Siwakorn Srisakaokul, Gene Kim, and Michael D Ernst. A format string checker for Java. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 441–444. ACM, 2014.

[8] Werner Dietl, Stephanie Dietzel, Michael D Ernst, Kivanç Muşlu, and Todd W Schiller. Building and using pluggable type-checkers. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 681–690. ACM, 2011.