# 論文の内容の要旨

# Practical User-Defined Analysis-Based Pointcuts in an Aspect-Oriented Programming Language

# アスペクト指向プログラミング言語における実用的な利用者定義の解析に基づくポイントカット

青谷知幸

関心事の分離 (SoC) の促進は、プログラミング言語の研究目標の１つである。分離された関心事はオブジェクト指向プログラミング言語におけるクラスやメソッド、ML や Haskell などの函数型プログラミング言語における函数やモジュールなどの、再利用可能なプログラム片に収められる。関心事がうまく分離できると、プログラム片の再利用性や保守性が向上するという利点がある。本研究では、アスペクト指向プログラミング言語について、プログラム解析に基づくポイントカットを実用化する技術を提案する。プログラム解析に基づくポイントカットは SoC を促進するために有効だと言われている。

アスペクト指向プログラミング（AOP）は、ロギングや並列化、メモ化、例外処理などの、クラスやメソッドではうまく分離できなかった関心事を、アスペクトと呼ばれる再利用性の高いプログラム片にまとめる技術である。ポイントカット・アンド・アドバイス（PA）機構は、AOP を実現する仕組みの１つとして注目され、Java 言語を拡張した AOP 言語 AspectJ をはじめとする実用的なプログラミング言語で採用されている。PA 機構では、ポイントカットの具体的なクラスやメソッドの名前への依存度が低いほど、アスペクトの再利用性が高い。名前の代わりにプログラム解析から得られる情報に基づくポイントカットは、そのようなポイントカットの１つであり、これまでにいくつかの実現手法が提案されてきた。

だが実用化を考えると、これらの手法には、

1. ポイントカット定義のためにポイントカット記述のためにそれぞれ独自の言語を使っているため AspectJ をはじめとする主要な AOP 言語との親和性が低く、実用的な処理系の実装方法が明らかで

ない
2. プログラム解析のためのライブラリが提供されていない
3. ポイントカットを自由に組み合わせられるような言語設計であるため、ポイントカットの利用者がポイントカットの定義者の意図に反する使い方をしてしまう

という問題点があった。

AspctJ における解析ポイントカットの実用化のために、本研究は (1) プログラム解析に基づくポイントカットの新しい記述方法と実用的な処理系の作成手法を確立するとともに、(2) プログラム解析のためのライブラリやツールをポイントカット定義のために利用する仕組みを開発し、(3) どのプログラムにおいても機能しないポイントカットの発見手法を提案した。以下に、それぞれの詳細を述べる。

（1）AspectJ の汎用的な条件（`if`）ポイントカットの活用する新しい記述方法を提案し、SCoPE と呼ばれるコンパイルの枠組みを開発することで実用的な処理系の実現した。AspectJ の標準的な処理系では、条件ポイントカットを利用してプログラム解析に基づくポイントカットを定義すると、プログラム解析が実行時に行われるため、非常に大きい実行時オーバーヘッドが生じる。SCoPE では条件ポイントカットのうちコンパイル時に評価できるものを自動的に発見し、アスペクトを織り込んだプログラムの上で評価することで、プログラム解析にかかる実行時オーバーヘッドの問題を解決した。

実用性の観点から比較すると、他のプログラム解析を利用したポイントカットの実現手法に対して、SCoPE の手法には次に述べる 2 つの利点がある。第 1 に、AspectJ の構文と意味を変更しないため AspectJ との高い親和性を持ち、プログラマは新たな言語要素の使い方や言語を覚える必要が無く、統合開発環境などの AspectJ 向けの便利なツールをそのまま利用できる。第 2 に、既にある AspectJ のコンパイラに対して条件ポイントカットの扱いのみを僅かに変更するだけで SCoPE を実現できるため、実装が容易である。実際に AspectJ の処理系の 1 つである abc コンパイラを拡張して SCoPE コンパイラを作成したところ、Scala と Java の合計約 1900 行のコードで実現できた。この SCoPE コンパイラは abc でコンパイルできるすべての AspectJ のプログラムをコンパイルする。本研究では、この SCoPE コンパイラを用いて、プログラム解析に基づくポイントカットを (1) 現実的な時間でコンパイルでき、(2)HotSpot コンパイルなどの標準的な Java 仮想機械での最適化が有効である場合にはコンパイルされたプログラムにプログラム解析が原因の実行時オーバーヘッドがないことを、４種類のプログラムをコンパイルして実行しその時間を比較することで明らかにした。

（2）SCoPE コンパイラの実装にあたって、AspectJ のプログラムからプログラム解析のためのライブラリやツールを利用するためのインターフェースを定義し、ASM、Soot の 2 つのライブラリと、Java で書かれたプログラムのバグ発見ツールである FindBugs の適用器を実装した。これらの適用器を適宜利用することによって、プログラマは、１からプログラム解析を記述することなくポイントカットを定義できる。`NullPointerException` が起こる場所を指定するポイントカットの記述のために、ポイントカットの実装者はインターフェースが提供するただ１つのメソッドを呼び出すだけでよいことを実例を示すことで明らかにした。

（3）ポイントカットに対して制約式を生成し、その制約式の中の矛盾を見つけ出すことで、どんなプログラムにおいても機能しないポイントカットをコンパイル時に発見する手法を提案した。どんなプログラムにおいても機能しないポイントカットを意図的に書くことはまれであるため、このようなポイントカットを見つけることは、バグのないプログラムを開発するために有効である。本研究の手法では、AspectJ の豊富なポイントカットについてポイントカット定義のみが与えられた状況でも、このようなポイントカットを発見できる。

本研究では Featherweight Java に PA 機構を導入して拡張した小さな計算体系 APFJ を定義し、矛盾の見つかった制約式を持つポイントカットが無意味であることを証明した。また、他の発見手法に比べて本研究の手法が有効であることを、無意味なポイントカットの例に対する発見可能性を比較することで示した。