様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　　元 年　　６ 月 １９ 日現在

機関番号：１５２０１
研究種目：挑戦的萌芽研究
研究期間：2016～2018
課題番号：１６Ｋ１２４１２

研究課題名（和文）意味論プラガブルなプログラム解析

研究課題名（英文）Semantics-Pluggable Program Analysis

研究代表者
神谷　年洋（Kamiya, Toshihiro）

島根大学・学術研究院理工学系・教授

研究者番号：７０４１５６６０

交付決定額（研究期間全体）：（直接経費）　　2,400,000円

研究成果の概要（和文）：本研究提案の目的は複数の意味論・開発技術を用いて開発されたソフトウェアを対象としたプログラム解析手法を実現することであった。その成果として、次の3つの手法を提案・実装し、初期的な実験により評価を行った。
(1) プログラム異なるバージョンの実行トレースを解析することによりバージョン間の差異を視覚化する手法、(2) ひとつのプログラムに少しだけ異なる入力データを与えて実行することにより、プログラム内のデータフローを解析し視覚化する手法、(3)異なるプログラミング言語で記述されたソースコードから、類似コード断片を検出する手法。

研究成果の学術的意義や社会的意義
近年のソフトウェア開発では、ソフトウェアは複数の開発技術(およびそれらが包含する意味論)の混成物として開発される。例えば、webアプリケーションであれば、JavaScriptやhttpといったインターネットに関連する技術、プログラムを記述するための種々のプログラミング言語、データを格納し問い合わせを行うためのデータベース等の技術が用いられたソフトウェアとなる。
ソフトウェアの解析はソフトウェアの不具合修正や品質の計測に用いられるが、従来の解析技術では、ソフトエアの内部に存在する開発技術の境界が壁となり、効果的な解析が行えない場面があった。本研究提案はその壁を乗り越えた解析を実現するものである。

研究成果の概要（英文）：The purpose of this research project was to establish a framework of program analysis, whose targets are software products developed with multiple semantics/technologies. The outcomes of the project were the following analysis methods with implementations and initial empirical evaluations; (1) visualization of differences between two versions of a program by analyzing their execution traces, (2) detection and a visualization method of data flow in a program by running the program with slightly different sets of input data, and (3) a detection method of similar code fragments written in different programming languages.

研究分野：ソフトウェア解析

キーワード：プログラム解析　プログラム理解　保守　デバッグ　動的解析

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９、ＣＫ－１９（共通）

１．研究開始当初の背景

コンピュータ・プログラムの保守作業は困難なタスクである。近年の**プログラムは複数の意味論・開発技術が混在した構成物**であることも原因の一つである。このような意味論には、例えば、プログラム自体を記述するためのプログラミング言語、プログラムが永続的なデータを扱うための仕掛けであるデータベースの構造やそのアクセスのための言語である SQL 等、ユーザーインターフェイスを記述するために用いられる HTML や JavaScript, UI フレームワーク等、アプリケーションのドメインに固有の技術や言語等があり、ひとつのプログラムの記述にこれら複数の技術が用いられる。

現在の**解析手法・技術の各々はこれらの意味論のひとつまたは一部しかサポートしない**ため、保守作業に携わる開発者は、たとえ解析ツールを利用したとしても、そのツールが関知しない意味論､サポートしない開発技術による因果関係や依存関係を手作業で補うことが必要となる。このような、意味論の混在による追跡性の低下は、今後も開発技術が進歩するにつれて(再利用ライブラリやコンポーネントの意味論の独立性が増して結合が粗になることにより)より深刻になると考えられる。

２．研究の目的

本研究提案は、このようなプログラムの保守作業(プログラムを理解する、不具合の原因を特定し修正する､保守性を向上させるために構造を変更する等)に携わる開発者をサポートすることを目的として、**プログラムに含まれている複数の意味論・開発技術を横断したプログラム解析の手法を実現**する。これにより、複数の意味論や開発技術を橋渡しした因果関係の鎖を追跡することを可能にする。

例えば、データベースを用いるプログラムにおいて、実行時にプログラムがユーザーから受け取った入力をデータベースに格納し、その後、プログラムがそのデータをデーテベースから取り出して、加工し、画面に出力するといったデータフローを解析するツールを実現する。このような解析ツールは、プログラムのデバッグにおいて、ユーザーが不具合であると認識した誤ったデータがプログラムやデータベースの間でどのように受け渡されたか理解し、不具合の原因を救命する作業で利用することができる。

別の例としては、複数のプログラミング言語を用いて開発されたプログラムにおいて、異なるプログラミング言語で記述されているが機能は類似しているコード断片を検出するツールを実現する。このような解析ツールは、あるプログラミング言語から別のプログラミング言語へと書き換えられている途上のプログラムがあり、そのようなプログラムに開発者が意図せずに重複したコードを作り込んでしまった状況において､開発者がプログラムの機能を修正する際に、そのような重複コードの存在を確認することで修正漏れをしないようにするために利用することができる。

３．研究の方法

本研究提案は、プログラムの実行トレースの解析を基本とする。ここで、実行トレースとは、解析対象となるプログラムを、特定の入力を与えて実行した際に、呼び出される手続きや、その手続の実引数・戻り値などを、プログラムの実行の開始から終了までの間記録したものである。

実行トレースを利用して、上述のような、プログラムに含まれる複数の意味論・開発技術を横断した解析を実現するために、次の 2 つの手法を提案・実装し、初期的な実験よる評価を行った。

(1) 異なる開発技術により開発されたプログラムの解析を可能にするために、解析対象のプログラムが異なるプログラミング言語で記述されていても、(ほぼ) 同じフォーマットで実行トレースを取得できる実行トレース収集手法およびツール(後述する研究成果において後述するが、これまでに実装および実験を行ったのはプログラミング言語 Java と Python に適用可能なものである)。これにより、(類似部分検出や差異の視覚化といった) 解析手法の同じ実装を異なるプログラミング言語で記述されたプログラムに対して適用すること、および、複数のプログラミング言語で記述された一つのプログラムに対して適用することが可能になる。

(2) 少しだけ異なる 2 つの入力データを与えてプログラムを実行し、それら 2 つの実行トレースの差異により、データフローを特定する手法。これは特に、プログラムの実行中にデータベースの問い合わせを行ったり､http などのプロトコルに乗せるためにデータを変換するようなプログラムに適用する。これらの処理はデータの構造や表現の変換を伴うため、通常のデータ

フローの追跡手法は利用できない（データにトレイトと呼ばれる印をつけたり、メモリ上のアドレスにより特定するといった手法が利用できない）状況においてデータフローの解析のために適用することを目的としたものである。

４．研究成果

(1) プログラム異なるバージョンの実行トレースを解析することによりバージョン間の差異を視覚化する手法

プログラムの修正によりどのように設計が変更されたかを分析することを目的として、2 つのバージョンを同じ入力に対して実行したものから収集された実行トレースを比較し、プログラムの実行中に呼び出される手続きがどのように変更されたかを特定して視覚化する。研究の方法の(1)に係る研究成果である。

本手法による解析においては、特定の機能の実装がソースコード内で移動したことだけではなく、それらの機能の間の関係（手続きの呼び出し関係の意味において）がどのように変化したかも視覚化することができる。これにより、特に、2 つのバージョンの間の修正がリファクタリング（プログラムの振る舞いを変えずに、すなわち、プログラムが入力するデータや出力するデータは変えずに、プログラムの構造を変更する）であったときに、変更の意図を、ソースコードの字面よりもより設計情報に近い（抽象度の高い）手続き呼び出しのレベルで分析することが可能となる。

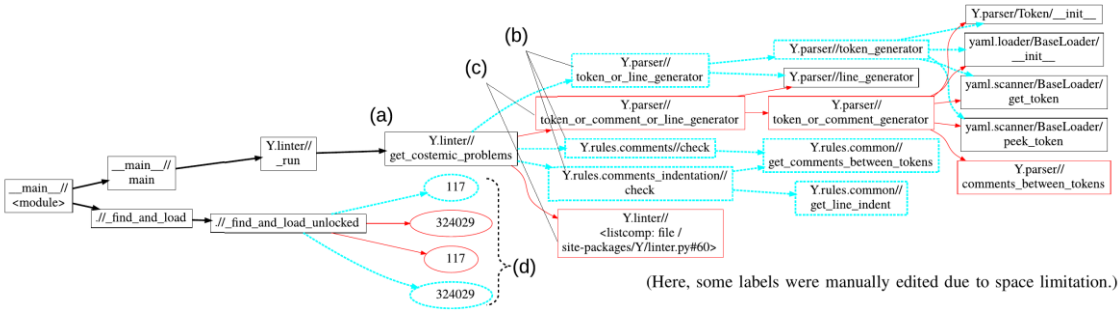

図１ プログラムのバージョン間の差異を手続き呼び出しの構造と違いとして視覚化したもの

本手法を実装し、リファクタリング前後での手続き呼び出しのレベルでプログラムの修正を視覚化し分析する実験を行った。図 1 に本手法の実装を小規模な Python で記述されたプログラムの 2 つのバージョンに適用した例を示す。

＜図の引用＞
Toshihiro Kamiya, “Code Difference Visualization by a Call Tree,” Proceedings of the 12th International Workshop on Software Clones (IWSC 2018), pp. 60-63 (Mar. 20, 2018).

(2) ひとつのプログラムに少しだけ異なる入力データを与えて実行することにより、プログラム内のデータフローを解析し視覚化する手法

プログラムが複数の開発技術により構成される状況においてもデータフローの追跡を可能にするために、解析対象となるプログラムに対して少しだけ異なる入力データを用意して与えて実行トレースを収集し、それらの実行トレース内に現れるデータの値がどのように異なるかを解析することによりデータフローを再構築する。研究の方法の(2)に係る研究成果である。

例えば、データベースや http などのプロトコルによる通信を含むようなプログラムの実行時には、プログラムを実行するプロセスとデータベースや通信先のブラウザとの境界においてデータがやり取りされるときに、データ構造や表現の変換が行われる。本手法はこのような状況においても、異なる入力に対する実行トレース中のデータの値の差異の解析によるデータフローの特定を可能にする。また、解析する実行トレースを 2 つではなく 3 つ、4 つと増やしていくことで、対象プログラムの実行ごとにランダムに異なってしまうデータの値（特定のデータがメモリ上に割り当てられるアドレス）と、入力の違いにより必然的に異なるデータの値を識別し、より精度が高いデータフローの特定を行えることも特徴である。

さらに、本解析手法にあわせて、データフローの視覚化手法も提案した。本視覚化手法は、プログラムの実行順に呼び出される手続きの中で、データフローを観察したいデータがどのように受け渡されているかを、実行トレースの一部を取り出してコンパクトに提示するものである。

図 2 データベースを介して渡されたデータフローを視覚化したもの

本解析および視覚化の手法を、プログラムの不具合の修正のために、不具合として表出されたデータ値を含むデータフローを分析するために利用する実験を行った。図 2 に、あるプログラムがユーザーからテキストデータを入力しデータベースに格納したあと、別のプログラムがデータベースからそのデータを読み出したデータフローを視覚化した結果を示す。

＜図の引用＞
神谷年洋, “データ値の差異とデータフローの視覚化によるデバッグ補助手法の提案”, SEA ソフトウェア・シンポジウム 2018 in 札幌 (SS2018), pp. 28-37 (2018-06-06).

(3)異なるプログラミング言語で記述されたソースコードから類似コード断片を検出する手法

レガシーマイグレーションを行っている開発プロセスで、あるプログラミング言語で記述されたソースコードを別のプログラミング言語へと書き換えていく途上において利用されることを想定した、異なるプログラミング言語で記述されたソースコードから類似コード断片を特定する手法である。研究の方法(1)に係る研究成果である。

本手法においいては、2 つの手続きの実行中に参照するデータの値が類似しているときに、それらの手続きが類似していると判定する。従来のように、2 つの手続きが同じプログラミング言語で記述されている場合に類似性を判定する場合には、コード断片が利用している標準ライブラリの API の類似性を用いることが可能である。それに対して、本手法が想定しているようなソフトウェア開発プロセスでは、異なるプログラミング言語により記述されたソースコードが混在しており、そのようなコード断片が利用している標準ライブラリの API の類似性は（プログラミング言語により標準ライブラリの API が全く異なるために）利用できない。本手法では、手続きが実行されている部分に相当する実行トレースに現れるデータの値の類似性を解析することにより、異なるプログラミング言語で記述されたコード断片の間での類似性の判定を可能にした。プログラミング言語が異なると利用される基本データ型の表現も異なる（ものがある）が、本手法ではデータ型の表現の差異を吸収しつつ比較する方法も取り入れている。

表 1 Java と Python で記述されたソースファイルからの類似コード断片検出結果

Python の手続き (c)	Java の手続き	V(c)	コサイン類似度
main	–	(55)	–
startMatch	←	55	0.528
startGame	←	35	0.478
playerMove	←	20	0.496
put	startGame,-1	17	0.305
display	←	16	0.567
compMove	println*,+1	14	0.546
checkForWin	startGame,-1	10	0.199
move	playerMoved	8	0.208
playerMoved	–	(7)	–
init	–	(6)	–
mark	playerMoved,-1	6	0.178
fixWeights	–	(4)	–
__init__	–	(1)	–

本手法を実装した類似コード断片検出ツールを、ごく小規模な Java プログラミング言語のプログラムと、それを Python に書き換えたプログラムとに適用する実験を行った。この実験では、手続きが十分に多くのデータ値を参照しているものについては、記述するプログラミング言語が異なっていても、2 つの手続きが類似していると判定することが可能であった。表 1 にこの実験での検出結果を示す。

＜表の引用＞
神谷年洋, “プログラミング言語横断類似コード断片検出ツールの試作”, SEA ソフトウェア・シンポジウム 2019 in 札幌 (SS2019), (2019-06-06).

５．主な発表論文等

〔雑誌論文〕（計２件）

① Toshihiro Kamiya, “Code Difference Visualization by a Call Tree,” Proceedings of the 12th International Workshop on Software Clones (IWSC 2018), pp. 60-63 (Mar. 20, 2018). 査読あり

② Toshihiro Kamiya, “Introducing Parameter Sensitivity to Dynamic Code-Clone Analysis Methods,” Proceedings of the 10th International Workshop on Software Clones (IWSC 2016), pp. 19-20 (Mar. 15, 2016). 査読あり

〔学会発表〕（計６件）

① 神谷年洋, “プログラミング言語横断類似コード断片検出ツールの試作”, SEA ソフトウェア・シンポジウム 2019 in 札幌 (SS2019), (2019-06-06). 査読あり

② 神谷年洋, “ソースコード推薦あるいは修正の情報源としての質問掲示板とソースコードレポジトリの比較”, 電子情報通信学会技術研究報告, vol. 118, no. 230, SS2018-23, pp. 31-36, (2018-10-5). 査読なし

③ 神谷年洋, “データ値の差異とデータフローの視覚化によるデバッグ補助手法の提案”, SEA ソフトウェア・シンポジウム 2018 in 札幌 (SS2018), pp. 28-37 (2018-06-06). 査読あり

④ 神谷年洋, “実行トレース間のデータの差異に基づくデータフロー解析手法の提案”, FOSE 2017 ライブ論文・ポスターセッション (2017-11-24). ソフトウェア工学の基礎 XXIV, 近代科学社, pp.217-218, 2017 年 11 月. 査読あり

⑤ 神谷年洋, ″実行トレース間のデータの差異に基づくデータフロー解析ツール″, 電子情報通信学会技術研究報告, Vol. 116, No. 136, pp. 55-60 (2017-07-19). 査読なし

⑥ 神谷年洋, “逆戻りデバッグ補助のための嵌入的スパイの試作,” 電子情報通信学会技術研究報告, Vol. 116, No. 127, SS2016-9, pp. 87-92 (2016-07-14). 査読なし

〔その他〕（計１件）

① 神谷年洋, ″コードクローン研究ふりかえり ～ ストロング・スタイルで行こう ～″, 第 19 回プログラミングおよびプログラミング言語ワークショップ (PPL2017) (2017-03-08).

※科研費による研究は、研究者の自覚と責任において実施するものです。そのため、研究の実施や研究成果の公表等については、国の要請等に基づくものではなく、その研究成果に関する見解や責任は、研究者個人に帰属されます。