科学研究費助成事業　　研究成果報告書

平成　２６　年　　６　月　　２　日現在

機関番号：　１２６０８
研究種目：　基盤研究(C)
研究期間：　2009～2013
課題番号：　２１５０００３３

研究課題名（和文）追記型・文字粒度の追跡子による，ソフトウェア追跡性の高精度な確保

研究課題名（英文）High-precision traceability by write-once/character-granularity tracers

研究代表者
権藤　克彦（Gondow, Katsuhiko）

東京工業大学・学術国際情報センター・教授

研究者番号：５０２６２２８３

交付決定額（研究期間全体）：（直接経費）　　3,500,000円、（間接経費）　　1,050,000円

研究成果の概要（和文）：ソフトウェア保守コストを下げるために，ソフトウェア追跡性を高めることが重要である．本研究では「追記式・文字粒度の追跡子」を用いた新しい手法を提案実装し，ソフトウェア追跡性を高めることを試みた．成果として，Unicode第14面を用いた追跡子の実装，ハッシュ値を用いたソースコードとドキュメントの一貫性保持方式の実装，厳格なアンサーセットで構成する追跡性リンクの事例集の作成を行った．また，追跡子の派生技術として，教育用コンパイラの研究などの成果も得た．

研究成果の概要（英文）：To reduce the cost of software maintenace, it is highly important to improve the software traceability. Our research proposed and implemented a new method using "tracer with write-once/character-granularity" to improve the software traceability. As a result, we implemented a tracer using the Plane 14 of Unicode and a consistency system using hash-value between source code and document, and provided a case study of traceability links as a strict answer set. We also proposed and implemented a new educational compiler and so on as derived technologies using tracers.

研究分野：　総合領域

科研費の分科・細目：　情報学・ソフトウエア

キーワード：　追跡性　追記式　文字粒度　ソフトウェア保守

１．研究開始当初の背景

研究開始当初，ソフトウェアの追跡性を確保する適切な手法が確立されていなかった．ソフトウェアの追跡性(software traceability)とは「ソフトウェアが満たすべき要求仕様，標準や規格，法令が何か，またそれが実装コードのどこに反映されたかを確認できる性質」である．例えば，建築物の構造計算ソフトウェアは，構造計算の手法や基準を定める建築基準法を満たす必要があり，ソフトウェアのどの部分が建築基準法のどの部分に対応しているのかを辿れるとき，このソフトウェアには追跡性が確保されていると言う．
莫大になるソフトウェア開発コストの大部分をソフトウェア保守費用が占める．ソフトウェアの追跡性の確保により，このソフトウェア保守費用を大幅に削減できるため，ソフトウェアの追跡性は非常に重要である．例えば，構造計算を行うソフトウェアのバグを修正した場合，どのコード部分が建築基準法のどの部分に対応しているかを追跡できれば，修正後のソフトウェアが建築基準法の要求を満たしていることを容易に確認できるからである．
既存研究や既存手法には，ソフトウェア・コンコーダンス，潜在的意味解析手法，ソースコード検索システム，文芸的プログラミングなどがあった．しかし，解析による追跡性の推測は不確実（精度が悪い），手動による追跡性情報の保守コストは高すぎる，追跡性情報自身がソフトウェアや文書の可読性を下げるという問題があった．

２．研究の目的

本研究では「追記型・文字粒度の追跡子」を提案・実装し，ソフトウェア追跡性の高精度な確保を目的とする．ポイントは次の通り：

(1) 特別な文字エンコーディング方式を開発して，plain text 中の各文字に追跡情報を埋め込み可能にする．
(2) 追跡子情報としては，識別番号，URL，作成者氏名などを追記方式で書き込む．削除や修正は不可．
(3) 追跡子情報は，通常は不可視にする．これにより可読性が落ちる問題を回避する．
(4) 追跡子情報は，最初だけユーザが手動で入力する．文字を修正しても追跡情報が残るので，変更・派生時に大部分の手動の保守が不要になる．

３．研究の方法

「追記型・文字粒度の追跡子」のための文字エンコーディング方式として，文字符号化方式の標準である ISO/IEC 2022(いわゆる ISO-2022)を用いる手法と，Unicode (UTF8)を用いる手法の両方を検討した．また，この検討した手法をベース技術として，次の３つの研究方針で研究を行った．
(1) 提案追跡子の機能を持つ簡易エディタを設計実装して，提案手法の実現可能性を明らかにすること．
(2) 提案追跡子の応用例として，ハッシュ値を用いた，ソースコードとドキュメント間リンクの矛盾検知方式のプロトタイプを試作すること．
(3) 追跡性リンクの有用性を示すための事例集として，厳格なアンサーセットで構成する追跡性リンクの事例集を作成すること．

４．研究成果

(1) Unicode 言語タグに基づく追跡子の実装

検討の結果，ISO/IEC 2022 に基づく追跡子方式の実現が困難であることが判明したため，Unicode に基づく追跡子方式を実現した．具体的には Unicode の第 14 面に配置されている言語タグ(RFC2482，RFC6082)を文脈性を廃止した形で追跡子として利用することにした．エディタなどのアプリケーションレベルでの実装となるが，Unicode 文字単位で追跡子か否かが判定可能な点で実装が容易であること，Unicode という同一のエンコーディングで複数言語を自然にサポートできるという大きな利点がある．ただし，簡単のため，ASCII コードを U+E0000～U+E00FF にマップして追跡子とすることにしたため，追跡子内で使用できる文字は ASCII コードのみとなった．
この追跡子を用いて，Java Swing 上に「追記型・文字粒度の追跡子」の機能を含んだ簡易エディタを試作した．また，「追記型・文字粒度の追跡子」の実装を Emacs のマイナーモードとしても実装した．後者の実装は言語タグ文字の入力，可視化・不可視化の制御なども可能となっており，実装規模は Emacs Lisp で約 300 行程度とコンパクトに実装できることを確認した．

(2) ハッシュ値を用いたソースコードとドキュメント一貫性保持方式の実装

追跡子の応用事例として，ハッシュ値を用いたソースコードとドキュメント一貫性保持方式の実現方法を検討し，プロトタイプを試作した．「追記型・文字粒度の追跡子」では通常の状態ではプログラマの目には見えない，隠れたテキストとして働くが，この隠れたテキストとして何を埋め込むべきかは自明ではない．ここでは外部のソフトウェアオブジェクトへの追跡性リンクが矛盾した状態(obsolete)になる問題を解決する方式として，ハッシュ値を用いる方式を提案した．具体的にはメソッドのボディ部分のテキストを対象としてハッシュ値を計算し，そのハッシュ値を見えない追跡子として埋め込む．リンク先のドキュメントなどが参照先として適切かどうかを確認せずに(つまりリンクを確認あるいは更新せずに)メソッドを書き

換えると，メソッドのハッシュ値と追跡子中のハッシュ値が矛盾するため，ツールによる自動的な警告が可能となる．
この方式を，Java 言語を対象として，Eclipse のプラグインとしてプロトタイプ実装した．実装の容易さのため，追跡子としては埋め込まず，Javadoc の@see タグが表現するリンクに対して，@hashcode タグ部分にハッシュ値を表現する方式をとった．これにより，コンパクトに実装できることや，ツールによる自動警告が可能であることを確認できた．

(3) 厳格なアンサーセットで構成する追跡性リンクの事例集の作成

追跡性リンクの有用性を示すための事例集として，厳格なアンサーセットで構成する追跡性リンクの事例集を作成した．ドキュメント理解の困難さ，膨大さ，分散性，頻繁な改訂などの観点から，対象として日本の税法を選択した．法人税，所得税，消費税の３法を対象に，独立した３０個のソースコード（それぞれ約１００行程度）を作成し，税法への追跡性リンクをコメントとして細粒度に付した．開発者の意図により正解・不正解が異なるリンクが多くあること，追跡性リンクの埋め込み自体に大きなコストがかかるが，細粒度に埋め込めたこと，埋め込んだ追跡性リンクの有用性を期待できることなどを確認した．

(4) 追跡子の派生技術の成果

本研究では，追跡子の派生技術として，多くの成果をあげた．追跡子技術の本質は，複数のものをつなぐリンクと，そのリンクをいかに効率よく辿るか，あるいはリンクをどのように利用するかである．各成果を以下に述べる．

・教育用コンパイラの研究（雑誌論文[3]，学会発表[1]）では，関連する様々なビュー（ソースコード，アセンブリコード，抽象構文木，スタックレイアウト等）をつなぐリンクとして，水平スライスを提案実装した．
・GPU を利用したポインタ解析の研究（発表論文[2]）では，ソースコード解析の基本であるポインタ解析技術として，GPU を用いて効率よくポインタ参照関係のリンクを辿る手法を提案実装した．
・誤解放を防ぐための新しい型修飾子 strict_lifetime の研究（雑誌論文[5]）ではコンパイラ最適化器の相互干渉により，使用中のデータへのリンクが消滅し，その結果，メモリの誤解放が生じる問題を扱った．strict_lifetime という新しい型修飾子を GCC に実装することでこの問題を解決した．
・バッファオーバーフローの研究（雑誌論文[4][6][7]，学会発表[2][3]）では，プロセス中のメモリの使用・未使用状況を効率よく追跡する手法を提案実装した．
・#include の事例研究（雑誌論文[8]）では，API（例えば，printf 関数）を使用する際に必要となる#include 文（例えば，#include <stdio.h>）の対応関係が，実際にどの程度間違っているのかを教科書の事例を対象として調査した．多くの誤った対応関係（例えば，#include 文の消し忘れ）が見つかり，追跡子の確保が必要という結果を得た．
・整数オーバーフローの未定義動作検出の研究（雑誌論文[1]）では，C 言語の未定義動作の脆弱性を効率よく発見するテストケースを提案し，従来のランダム手法よりも 36.7% 多くの問題を発見することを示した．未定義動作を記述してしまった際に，プログラム動作の症状として顕在化させること，つまりバグから症状への追跡性を高める技術は重要であり，本研究はその一助となっている．

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計８件）

[1] 整数オーバーフローの未定義動作検出に対する整数境界値の定量的評価，森川知哉，荒堀喜貴，権藤克彦，コンピュータソフトウェア（レター論文）[31]，No.1，pp.103-109 (2014)，査読あり，https://www.jstage.jst.go.jp/article/jssst/31/1/31_1_103/_pdf

[2] GPU を利用したポインタ解析の実装と評価，深谷敏邦，権藤克彦，コンピュータソフトウェア（レター論文）[29]，No.3，pp.70-76 (2012)，査読あり，https://www.jstage.jst.go.jp/article/jssst/29/3/29_3_70/_pdf

[3] 権藤克彦，福安 直樹，荒堀 喜貴：ネイティブアセンブリコードを出力する教育用コンパイラ (XCC)と，水平スライスが可能な可視化ツール (MieruCompiler)，電子情報通信学会論文誌，vol.J95-D，No.5，pp.1225-1241 (2012)，査読あり，http://ci.nii.ac.jp/naid/110009444739

[4] 荒堀喜貴，権藤克彦，前島英雄：競合回避機構を備えた高互換かつ高精度な境界検査手法，情報処理学会論文誌 53(3)，pp.1150-1165 (2012-03-15)，査読あり，http://ci.nii.ac.jp/naid/110008802673

[5] 鮎川力也，権藤克彦，荒堀 喜貴：誤解放を防ぐための新しい型修飾子 strict_lifetime，電子情報通信学会論文誌，vol.J95-D，no.2，pp.217-224 (2012)，査読あり，http://ci.nii.ac.jp/naid/110009328418

[6] 荒堀喜貴，権藤克彦，前島英雄：広範な実用 C プログラムに適用可能かつ高精度な動的境界検査ツール，電子情報通信学会論文誌，vol.J93-D，no.10，pp.1851-1865 (2010)，査読あり，http://ci.nii.ac.jp/naid/110007730846

[7] 荒堀喜貴，権藤克彦，前島英雄：C プログラムの割込み競合の動的検出法，情報処理学会論文誌，vol.51，no.9，pp.1816-1831 (2010)，査読あり，http://ci.nii.ac.jp/naid/110007970782

[8] 権藤克彦，冨永 和人：事例研究：ツール「簡単#include 検査君」とその経験，コンピュータソフトウェア（レター論文）[27]，No.2，pp.93-99 (2010)，査読あり，https://www.jstage.jst.go.jp/article/jsst/27/2/27_2_2_93/_pdf

〔学会発表〕（計３件）

[1] K. Gondow, N. Fukuyasu, Y. Arahori: MieruCompiler: Integrated Visualization Tool with ``Horizontal Slicing'' for Educational Compilers, 41st ACM Technical Sympo. on Computer Science Education (SIGCSE 2010), pp.7-11, (2010.3.11), 米国ミルウォーキー，DOI: 10.1145/1734263.1734268

[2] Y. Arahori, K. Gondow, H. Maejima: Cache-Based Bounds Checking for Multi-Threaded C Programs, 21st IASTED Int. Conf. Parallel and Distributed Computing Systems (PDCS 2009), CD-ROM No.668-019, Sep., (2009.11.3), 米国ボストン

[3] Y. Arahori, K. Gondow, H. Maejima : TCBC: Trap Caching Bounds Checking for C, 8th Int. Conf. Dependable, Autonomic and Secure Computing (DASC-09), pp. 49-56, Dec. (2009.12.12)，中国成都，DOI: 10.1109/DASC.2009.80

〔図書〕（計０件）

〔産業財産権〕
○出願状況（計０件）
○取得状況（計０件）

〔その他〕

６．研究組織
(1)研究代表者

権藤 克彦 (GONDO KATSUHIKO)
東京工業大学・大学院情報理工学研究科・教授
研究者番号：50262283
(2)研究分担者
(3)連携研究者