1K-05

x86-64のネイティブコードを対象にした逆コンパイラのコード復元性能の評価方法の検討

栃原 大器† 神崎 雄一郎†

† 熊本高等専門学校

1 はじめに

IDA [1] や Ghidra [2] など，逆コンパイラ (逆コンパイルを行う機能) を備えたバイナリ解析ツールが近年では多く存在する．逆コンパイルとは，機械語のコードやバイトコードを人間が読みやすい高水準言語のプログラムに変換することである．逆コンパイラはマルウェア解析などに活用される一方で，機密性の高いコードを持つソフトウェアに対する逆コンパイラを用いた不正な解析行為は，ソフトウェア開発者にとって脅威となり得る．そのため，近年の逆コンパイラを用いたソフトウェア解析が実際にどの程度有用かを把握する手段が求められる．

逆コンパイラに関する研究として，人間が逆コンパイルするプロセスに関する分析 [3] や，Java バイトコードを対象とした逆コンパイラの性能評価 [4] などが行われている．一方，IDA など，近年発展している x86-64 のネイティブコードに対応する逆コンパイラの性能を系統的に評価する方法は十分に明らかになっていない．そこで本研究では，x86-64 のネイティブコードを対象にした逆コンパイラのコード復元性能を評価する方法を検討する．ここでコード復元性能は，与えられたネイティブコードを，元来のソースコードと同じ意味を持つ高水準言語のプログラムにどの程度正確に変換できるかを指す．提案方法では，逆コンパイラが出力する疑似コードをコンパイル可能なプログラムに変形し，そのプログラムが元来のソースコードとどの程度類似しているか，また，元来どおりの動作をするか，といった観点からコード復元性能を評価する．最近の逆コンパイラとして IDA を対象にコード復元性能を評価する実験を行い，得られた結果について考察する．

2 提案方法

提案方法では，逆コンパイルされたコードが (1) 元来のコードとどの程度類似しているか，(2) 元来の入出力関係を保っているか，の 2 点から逆コンパイラのコード復元性能を評価する．ここでは，x86-64 の機械語コードから C 言語の疑似コードを生成できる逆コンパイラを評価の対象とし，機械語コードの元となる C 言語のソースコードが入手可能であること，また，逆コンパイルを関数の単位で行うことを前提とする．

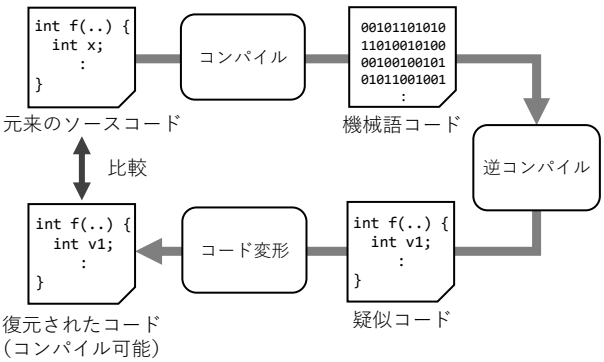

図 1 復元されたコードと元来のコードを比較する流れ

図 1 に示すのは，逆コンパイルされたコードと元来のコードを比較し，類似性を評価する流れである．まず，対象となる関数のソースコードをコンパイルし，機械語コードを生成する．その機械語コードを逆コンパイラによって逆コンパイルし，疑似コードを取得する．逆コンパイラはユーザの理解を助けることが目的であるため，生成された疑似コードは必ずしもコンパイル可能な状態であるとは限らない．そこで，疑似コードで用いられる型の定義の追加など，プログラムの意味に影響を与えない軽微な修正を必要に応じて (人手によって) 疑似コードに加え，コンパイル可能な状態にしたコード（以降,「復元されたコード」と呼ぶ）を取得する．そして，復元されたコードと元来のコードとを比較し，C 言語やアセンブリのレベルで類似度を測定する．

復元されたコードが元来の入出力関係を保つかという点 （上記の (2)）は，元来の全体のソースコードのうち，該当する関数を復元されたコードに置き換えて得た実行可能コードに対して，一定の入力に対する出力が元来と同じになるかをテストすることで評価する．

An Attempt to Evaluate the Ability of Decompilers for x86-64 Native Code to Recover Source Code
Taiki Tochihara†, Yuichiro Kanzaki†
† National Institute of Technology, Kumamoto College

表 1　復元されたコードの動作テストの結果・アセンブリ命令数・元来のコードとの類似度

プログラム名	難読化なし			難読化あり (Encode Arithmetic)		
	動作テスト	命令数 復元後（元来）	類似度	動作テスト	命令数 復元後（元来）	類似度
change.c	成功	88 (77)	0.798	失敗	1,662 (793)	0.473
hanoi.c	成功	42 (38)	0.776	成功	100 (99)	0.903
lucas.c	成功	121 (122)	0.984	成功	352 (376)	0.753
zeta.c	成功	104 (105)	0.903	成功	159 (164)	0.788

3　実験

2 章で述べた 2 つの観点からコード復元性能を評価する実験を行い，結果について考察する．コンパイラは GCC (version 4.8.5)，逆コンパイラは IDA (version 8.3) [1] を用いる．実験対象のプログラムとしては，文献 [5] に関連する Web ページ[1]に掲載されているもののうち，小銭で払うパターンの数を計算する change.c，ハノイの塔の問題を解く hanoi.c，メルセンヌ数が素数かを判定する lucas.c，Riemann のゼータ関数を計算する zeta.c の 4 つを用いる．各プログラムは，main 関数を含んだ複数の関数で構成されており，main 関数ではない主要な関数 1 つを評価対象とする．また，各プログラムについて，コード難読化ツール Tigress (version 3.1) [2]の Encode Arithmetic の難読化（整数演算を複雑な表現に変形する難読化）を適用したものを作成し，実験対象に含める（実験対象のプログラムは計 8 つとなる）．なお，今回は機械語コードからのシンボル情報の除去は行わないものとする．復元されたコードと元来のコードとの類似性評価は，アセンブリのレベルで行う．具体的には，評価対象となる関数の 3-gram アセンブリ命令列（GNU Assembler 形式の 3-gram オペコード列）の出現頻度をそれぞれのコードから取得し，両者の間のコサイン類似度を測定する．

図 1 の「コード変形」に相当する処理として行ったのは，IDA 付属のヘッダファイルを取り込む指令の追記，関数の呼び出し規約の修正，評価対象の関数で用いられているグローバル変数の宣言の追記などであった．実験の結果を表 1 に示す．表 1 では，各実験対象の復元されたコードについての動作テストの結果，評価対象の関数のアセンブリ命令数，元来のコードとの類似度を示している．命令数の項目では，元来のコードの命令数を併記している．

動作テストの結果としては，難読化された change.c を復元したコードのみ，失敗となった．このコードは，一定の値を超える入力値に対する出力が元来のものと異なっていた．このコードは命令数が特に多く，元来のコードとの類似度も 0.473 と低い．難読化を適用しない場合には動作テストが成功していることを考えると，難読化によって正確な逆コンパイルが困難になった可能性があると考える．類似度については，動作テストに失敗したものを除いては 0.7 を超える値となっている．難読化なしの lucas.c や zeta.c，難読化ありの hanoi.c を復元したコードは 0.9 を超える高い値となっており，これらのコードは特に，それぞれの元来のコードと似たアセンブリ命令列の構成になっているといえる．

復元されたコードのほとんどが動作テストに成功し，元来のコードとの類似度が高いものが多く存在していることから，IDA の逆コンパイラの高い復元性能がうかがえる．他の逆コンパイラと評価結果を比較することは，今後の重要な課題となる．

[1]algo-c: https://github.com/okumuralab/algo-c/
[2]Tigress: https://tigress.wtf/

4　おわりに

本研究では，x86-64 のネイティブコードを対象にした逆コンパイラのコード復元性能の評価方法を検討した．また，提案方法を用いて IDA の逆コンパイラのコード復元性能を評価する実験を行い，結果を確認した．今後の課題として，評価対象とする逆コンパイラや類似度の計測方法を増やすなど，規模を拡大した実験を行うことが挙げられる．

謝辞　本研究は，JSPS 科研費 JP22K11986 の助成を受けたものである．

参考文献

[1] Hex-Rays, “IDA Pro,” https://hex-rays.com/ida-pro/.

[2] “Ghidra,” https://ghidra-sre.org/.

[3] K. Burk, F. Pagani, C. Kruegel, and G. Vigna, “Decomperson: How humans decompile and what we can learn from it,” 31st USENIX Security Symposium (USENIX Security 22), pp.2765–2782, Aug. 2022.

[4] N. Harrand, C. Soto-Valero, M. Monperrus, and B. Baudry, “The strengths and behavioral quirks of Java bytecode decompilers,” 19th International Working Conference on Source Code Analysis and Manipulation, pp.92–102, 2019.

[5] 奥村晴彦，[改訂新版] C 言語による標準アルゴリズム事典，技術評論社，2018.