7H-04

類似コード検出に向けた中間言語の視覚化

半澤　順一†　　佐藤　喬†　　小宮　常康‡
†東京都立産業技術高等専門学校　　‡電気通信大学 大学院情報システム学研究科

1　背景

ソフトウェアの長期開発や多人数開発では，機能が重複した類似コードを書いてしまう場合がある．類似コードが存在すると，ソフトウェアの保守コスト増大による品質低下やソースコードをコンパイルした後の実行ファイルサイズの肥大化に繋がる．類似コードを統合することで，実行ファイルサイズの肥大化を防ぐことが可能となる．しかし，高級言語のソースレベルでは開発者によって選択する構文や変数名等，書き方の違いがあるため，類似コードを発見するためには詳細な解析をしなければならない．そこでコンパイラの中間言語への変換を行い，この違いを低減した命令を得ることで，解析を行いやすくする．中間言語とはコンパイルの過程で生成されるコードを指し，高級言語よりも命令が単純な言語である．

本研究では，類似コードの発見の補助を行うことを目的とする．人間の目で類似コードを発見するために，ソースコードの中間言語のテキスト情報で色を決め，対応するソースコードに色を付け，ソースコードの流れを俯瞰できるように視覚化する．まず，中間言語の単一命令に対して色付けを行う予備実験を行った．このとき対象とする高級言語はC言語とした．予備実験を行った結果，プログラム全体の把握が難しいため，視覚化する際に命令をグループ化する必要があることがわかった．そこで，視覚化するためのグループ化の手法を提案し，グループを一次元の色の並びとして表示するシステムを開発する．

2　関連研究

Sargsyan らの研究 [1] では，中間言語から生成するプログラム依存グラフを用いて類似コードを検出を行っている．実験として，Mozilla Firefox のコードを 8 時間で分析し, 類似が疑われるコードを 98 個検出し，そのうち 91 個が類似コードであったとしている．本研究と同様に中間言語を用いて類似コードを検出しているが，本研究では中間言語を視覚化した結果を開発者自身が俯瞰し，類似コードの発見を行う点が異っている．

3　本システムの概要

本研究ではコンパイラ基盤の LLVM で生成した中間言語レベルで類似コードを発見し，結果をソース言語レベルで表示することで類似コードの発見の支援を行う．LLVM は変換元に対応するフロントエンドと変換先に対応するバックエンドが中間言語を境界に分かれているため，モジュールによる再利用性があり，様々な高級言語への対応が可能となる特徴をもつ．

中間言語はコンパイル中に生成されるコードを指し，高級言語よりも命令が単純なコード列である．さらに LLVM では変数定義が唯一となるように設計されているため，変数への格納や比較といった一つの動作ごとに変数が完結しているという特徴がある．そのため，動作ごとに類似比較が行え，高級言語で考慮する問題が低減される特徴を持つ．

中間言語を視覚化する流れを図 1 に示す．視覚化する流れは，生成された中間言語を本システムに通し, 命令に対応した色を 1 次元上に並べた出力ファイルの生成を行う．視覚化した色の並びから人間の目によって類似コードを発見する．

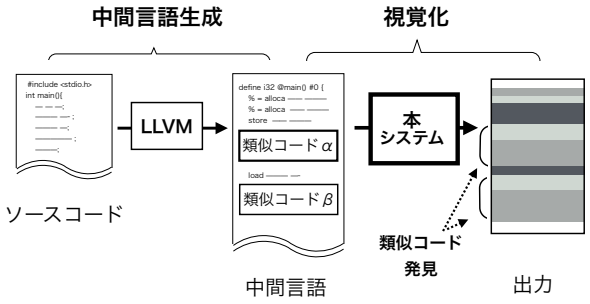

図 1: 中間言語を視覚化する流れ

4　予備実験

既存手法はプログラムの類似性を判断するために, PDG(Program Dependence Graph) を用いていたが, PDG を生成する時間がかかる．本研究では, コンパイラが生成した中間言語のテキスト情報からオペランドを無視した命令名のみで視覚化し，人間の目によって類似パ

Visualization of Intermediate Representation for Similar Code Detection
†Junichi HANZAWA　Takashi SATO　‡Tsuneyasu KOMIYA
†Tokyo Metropolitan College of Industrial Technology
‡Graduate School of Information Systems, The University of Electro-Communications

ターンを発見する手法をとる．そのため，PDG を生成，解析する時間を短縮できる．

中間言語の単一命令を色と対応させ，一次元の色の並びを作成する．そのプログラムの制御構造の把握が可能であるかの予備実験を行った．対象プログラムをリスト 1 に，実験結果を図 2 に示す．

リスト 1: C プログラム

```
1  int  main(void){
2    int  x = 10;
3    if ( x == 7 )
4      printf (”seven.”) ;
5    else  if ( x == 5 )
6      printf (”five .”) ;
7    else
8      printf (”it ’s other”);
9  }
```

図 2: 出力結果

図 2 では，色が繰り返し出力している部分が存在するが，これは変数の値を読み込むたびに出力されるため，構造の把握はできない．また大規模なプログラムを扱った場合，中間言語の命令の種類も増加するため，視覚化したファイルから人間の目でプログラム構造の流れを追うことは難しい．そこで，命令同士をグループ化して，プログラム全体を把握しやすくなるか検討する必要がある．

5 グループ化手法の提案

予備実験結果から命令のグループ化が必要であることがわかった．そこで同じカテゴリの命令のグループ化，ベーシックブロック単位でのグループ化，ベーシックブロック内に出現した複数命令のグループ化の 3 種類の手法の提案を行う．

5.1 同じカテゴリの命令のグループ化

LLVM の中間言語には多様な命令が存在する．それら命令は 8 種類のカテゴリに分類することができる．以下の表 1 にカテゴリ名と代表的な命令の例を記載する．

表 1: 命令のカテゴリとその例

カテゴリ名	代表的な命令例
ターミネータ命令	ret, br, ...
2 項命令	add, mul, ...
ビット 2 項命令	and, or, ...
ベクトル命令	insertelement, ...
集合命令	extractvalue, ...
メモリアクセス命令 /アドレス命令	alloca, load, store, ...
変更命令	trunc to, sext to, ...
他の命令	icmp, call, ...

表 1 に示したカテゴリを同一の色で色を付けることで，似た意味をもつ命令を同系色として視覚化することができるため，予備実験の際よりもコード全体の把握がしやすくなる．しかし，大規模プログラムを適用した場合には命令の数はグループ化していないため，予備実験同様に視覚化した際の出力サイズはプログラムに比例して増加する．

5.2 ベーシックブロック単位でのグループ化

カテゴリごとのグループ化では，出力サイズは小さくならない．そこで LLVM でベーシックブロックと呼ばれる命令をまとめた制御ブロックに注目する．このベーシックブロックを一つのグループとして考え，ブロック内の命令列が同一の場合に同一の色をつける．ベーシックブロック単位ではグループ化する粒度が粗いため，ベーシックブロック内に特徴のある複数命令が存在する場合でも，前後の命令と同一のブロックとして形成されてしまう．そのため，複数命令同士の特徴を発見できない可能性がある．

5.3 ベーシックブロック内に出現した複数命令のグループ化

ベーシックブロック単位でグループ化をした場合，特徴のある複数命令が前後の命令と同一のブロックとして形成されてしまい，命令同士の特徴の発見が難しくなる．そこで，グループ化する粒度を細かくするために，ベーシックブロック内でグループ化を行う．粒度を細かくすることで，発見が困難であった命令同士の特徴を発見できる可能性が増加する．

6 まとめと今後の予定

ソフトウェア内に類似コードが存在すると，コードサイズの肥大化に繋がる．類似コードを統合することでコードサイズの削減が可能となる．本研究では，類似コードの発見の補助を行うことを目的とする．

手法として，LLVM の中間言語に変換し，視覚化を行い，類似コードの検出に向けた視覚化ツールを開発する．予備実験として，中間言語の単一命令に対して，色付けを行ったところ，命令のグループ化が必要であるという結論に至った．そこで，本論文では，カテゴリごと，ベーシックブロックごと，複数命令ごとの 3 種類のグループ化手法の提案を行った．

今後予定として，提案したグループ化について実装し，どの程度の粒度が適切かを評価していく．

参考文献

[1] Sevak Sargsyan, Shamil Kurmangaleev, Andrey Belevantsev, Hayk Aslanyan, Artiom Baloian, Scalable code clone detection based on semantic analysis, ISP RAN, vol.27, pp.39–49, 2015.